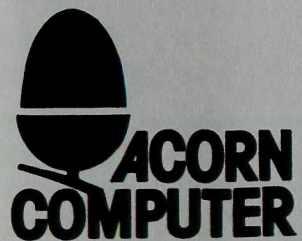




User's manual



Acorn Computer Limited, 4a Market Hill, Cambridge,
CB2 3MJ, Telephone (0223) 312772

CONTENTS

PART 1

CHAPTER 1: AN INTRODUCTION TO THE BINARY NUMBER SYSTEM

- 1.1 BINARY NUMBERS
- 1.2 LOGICAL MANIPULATIONS
- 1.3 ARITHMETIC MANIPULATIONS
- 1.4 BINARY CODED DECIMAL (BCD) ARITHMETIC

CHAPTER 2: WELCOME TO THE MACHINE

- 2.1 HOW THE ACORN MICROPROCESSOR WORKS
- 2.2 THE MONITOR COMMANDS M, ↑, ↓.
- 2.3 AT LAST, A PROGRAM,
 - 2.3.1 ASSEMBLY LANGUAGE, MACHINE LANGUAGE, THE INSTRUCTIONS LOAD, STORE and JUMP
 - 2.3.2 ENTERING A PROGRAM, THE GO COMMAND
 - 2.3.3 INSTRUCTIONS JMP, JSR
 - 2.3.4 LOGIC INSTRUCTIONS ORA AND EOR.
 - 2.3.5 ARITHMETIC OPERATIONS: ADC, SEC, CLC

CHAPTER 3: INSIDE THE 6502

- 3.1 THE ACCUMULATOR, PROGRAM COUNTER, STATUS REGISTER
- 3.2 THE STACK POINTER,
- 3.3 THE INTERNAL REGISTERS X & Y
- 3.4 MAKING OUR PROGRAM 'FRIENDLY'

CHAPTER 4: THE REMAINDER OF THE INSTRUCTION SET

- 4.1 BRANCHES
- 4.2 INDEXING
- 4.3 INDIRECTION
- 4.4 READ – MODIFY – WRITE INSTRUCTIONS
- 4.5 MISCELLANEOUS REMAINING INSTRUCTIONS

CHAPTER 5: ACORN HARDWARE

- 5.1 CHIP LAYOUT AND BUS
- 5.2 RESET, INTERRUPT REQUEST AND NON-MASKABLE INTERRUPT
- 5.3 6502 INTERNAL ARCHITECTURE
- 5.4 PROMS, EPROMS, RAM, RAM I/O
- 5.5 THE KEYBOARD AND TAPE INTERFACE
- 5.6 POWER SUPPLY

CHAPTER 6: FIRMWARE

- 6.1 THE TAPE STORE AND LOAD
- 6.2 THE BREAKPOINT AND RESTORE COMMAND
- 6.3 THE SINGLE STEPPING FACILITY
- 6.4 THE MONITOR LISTING

PART 2

APPLICATION PROGRAMS

APPENDICES

APPENDIX A: 64 CHARACTER ASCII ON ACORN'S 7 SEGMENT DISPLAY

APPENDIX B: INSTRUCTION SET

APPENDIX C: HEXADECIMAL TO DECIMAL CONVERSION TABLE

APPENDIX D: ACORN MONITOR ADDRESS INFORMATION

GLOSSARY

PART 1

CHAPTER 1: AN INTRODUCTION TO THE BINARY NUMBER SYSTEM

1.1 BINARY NUMBERS:

NUMBERS IN EVERY DAY USE ARE WRITTEN IN THE DECIMAL SYSTEM, THAT IS, TO THE NUMBER BASE 10. A POSITIONAL NOTATION IS USED REPRESENTING ONE '100's; TWO '10's & EIGHT '1's AS THE SYMBOL 128. THE RIGHTMOST (i.e. LEAST SIGNIFICANT) DIGIT IS IN THE "UNITS" COLUMN, THE 2 IN THE "TENS" COLUMN, THE 1 IN THE "HUNDREDS" COLUMN, AND THE VALUE OF THE SYMBOL '128' IS EVALUATED AS $1 \times 100 + 2 \times 10 + 8 \times 1 = 128$. SIMILARLY '1024' IS EVALUATED AS $1 \times 1000 + 0 \times 100 + 2 \times 10 + 4 \times 1 = 1024$, WHICH IS MORE CONVENIENTLY WRITTEN AS $1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 = 1024$, USING THE MATHEMATICAL SHORTHAND FOR $1000 = 10 \times 10 \times 10 = 10^3$, AND THE CONVENTION "ANY NUMBER TO THE POWER ZERO IS 1" TO GIVE A CONSISTENT METHOD OF EVALUATING SUCH SYMBOLS.

SO 1024

CAN BE WRITTEN IN COLUMNS

3	2	1	0
1	0	2	4

AND EVALUATED AS $1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$ TO THE BASE 10.

TO THE BASE 8, 1024 WOULD MEAN $1 \times 8^3 + 0 \times 8^2 + 2 \times 8^1 + 4 \times 8^0$ WHICH IS THE DECIMAL NUMBER 532.

TO THE BASE 16, 1024 WOULD MEAN $1 \times 16^3 + 0 \times 16^2 + 2 \times 16^1 + 4 \times 16^0$ WHICH IS THE DECIMAL NUMBER 4132.

TO DISTINGUISH THE BASE TO WHICH A NUMBER IS WRITTEN WE'LL WRITE ITS' BASE AFTER IT AS A SUBSCRIPT: 1024_{10} AND NOW WE CAN WRITE

$$\begin{array}{rcl} 1024_8 & = & 532_{10} \\ 1024_{16} & = & 4132_{10} \end{array}$$

$$10000000_2 = 128_{10}$$

JUST AS BASE TEN HAS THE NAME 'DECIMAL', BASE SIXTEEN HAS THE NAME 'HEXADECIMAL', BASE EIGHT HAS THE NAME 'OCTAL' AND BASE TWO 'BINARY'. THESE FOUR BASES ARE IN COMMON USE WITH MODERN COMPUTERS, ESPECIALLY HEXADECIMAL (HEX) AND BINARY. CONVERSION BETWEEN BINARY, OCTAL & HEX NUMBERS IS VERY SIMPLE. SINCE THEY ARE ALL POWERS OF TWO, NUMBERS JUST NEED DIVIDING UP:—

$$\begin{aligned} 10000000_2 &= 110001100001_{16} = 80_{16} \\ &= 10101100011000_8 = 200_8 \end{aligned}$$

— EACH HEX DIGIT IS FOUR BINARY DIGITS (BITS) & EACH OCTAL DIGIT IS 3 BITS.

OCTAL DIGITS ARE 0, 1, 2, 3, 4, 5, 6, 7.

HEX DIGITS ARE 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F A....F ARE USED INSTEAD OF 10....15 TO ALLOW UNRESTRICTED USE OF THE POSITIONAL SYSTEM.

- PROGRAM COUNTER: 16 BIT REGISTER WHICH CONTAINS THE ADDRESS OF THE INSTRUCTION BEING EXECUTED. DURING EXECUTION THE PROGRAM COUNTER IS STEPPED UP TO POINT AT THE NEXT INSTRUCTION.
- PROM: PROGRAMMABLE READ ONLY MEMORY. THIS TYPE OF MEMORY ARRIVES BLANK. IT CAN BE PROGRAMMED BY THE USER WITH THE HELP OF A SPECIAL PROM BLOWER. ONCE THIS PROGRAM HAS BEEN PUT IN, IT CANNOT BE CHANGED.
- RAM: RANDOM ACCESS MEMORY. THIS IS THE STANDARD READ/WRITE MEMORY. DATA (AND PROGRAMS) ARE LOST WHEN THE POWER IS SWITCHED OFF.
- REGISTER: STORAGE LOCATION IN THE MICROPROCESSOR ITSELF. THERE ARE INTERNAL REGISTERS A, X, Y, PC, S, P.
- ROM: READ ONLY MEMORY. THIS IS MEMORY THAT HAS A PROGRAM PUT IN DURING PRODUCTION. THIS PROGRAM CANNOT EVER BE CHANGED, IT CAN ONLY BE READ.
- STORE: TRANSFERS DATA FROM AN INTERNAL REGISTER TO MEMORY.
- XTAL: THE CRYSTAL IN THE ACORN OSCILLATES AT 1 MHZ. i.e. ONE MILLION TIMES A SECOND. IT DOES THIS WITH GREAT ACCURACY. SO YOU CAN BUILD A CLOCK FROM YOUR ACORN.

CONVERSION TABLE

HEX	DECIMAL	OCTAL	BINARY
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111
10	16	20	10000
20	32	40	100000
40	64	100	1000000
64	100	144	1100100
80	128	200	10000000
100	256	400	100000000

THE ACORN MICROPROCESSOR IS DESIGNED TO DEAL WITH 8 BITS AT A TIME. THE COLLECTION OF 8 BITS IS GIVEN THE SPECIAL NAME 'BYTE', AND IS NORMALLY WRITTEN IN HEXADECIMAL OR BINARY. A BYTE THUS IS $0...FF_{16}$; $0...11111111_2$ OR $0...255_{10}$. THE MICROPROCESSOR CAN CARRY OUT LOGICAL AND ARITHMETICAL MANIPULATIONS ON BYTES.

1.2 LOGICAL MANIPULATIONS

THE MICROPROCESSOR CAN IMMEDIATELY CARRY OUT THE LOGICAL AND, EXCLUSIVE - OR & OR FUNCTIONS ON ALL 8 BITS SIMULTANEOUSLY, USING THE FOLLOWING TRUTH TABLES FOR EACH BIT (SYMBOL 'b')

AND (\wedge)

b ₁	b ₂	result
0	0	0
0	1	0
1	0	0
1	1	1

EXCLUSIVE - OR (∇)

b ₁	b ₂	result
0	0	0
0	1	1
1	0	1
1	1	0

OR (\vee)

b ₁	b ₂	result
0	0	0
0	1	1
1	0	1
1	1	1

EXAMPLE
OPERANDS

00111100

01011010

AND

(OPERATOR)

00011000 RESULT

00111100

01011010

E-OR

01100110

00111100

01011010

OR

01111110

1.3 ARITHMETIC MANIPULATIONS

BINARY ADDITION
WITH CARRY OUTPUT

b ₁	b ₂	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

BINARY ADDITION WITH CARRY FROM RIGHT

b ₁	b ₂	INPUT CARRY	SUM	OUTPUT CARRY TO LEFT
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

EXAMPLE: 00111100 $3C_{16}$ 60_{10}
 $01011010+$ $5A_{16} +$ $90_{10} +$
 10010110 96_{16} 150_{10}

IN ORDER TO MAKE LONGER ADDITIONS EASIER TO PROGRAM, THE MICROPROCESSOR HAS A CARRY BIT (FLAG). AT THE START OF AN ADDITION THIS IS TREATED AS THE INPUT CARRY, AND AT THE END IT RECEIVES THE CARRY OUT FROM THE SUM AT BIT 7: ASSUMING WE HAVE A CARRY INPUT:

11000011 $C3_{16}$ 195_{10}
 10100101 $A5_{16}$ 165_{10}
 $\quad \quad \quad [1] +$ $1_{16} +$ $1_{10} +$
CARRY OUT $[1] 01101001$ 169_{16} 361_{10}

SUBTRACTION OPERATES IN A SIMILAR MANNER, EXCEPT THAT THE CARRY (OR BORROW) FLAG OPERATES UPSIDE DOWN: A 0 CARRY FLAG IS TREATED AS REPRESENTING A BORROW FROM THE PREVIOUS STAGE:

11111111 FF_{16} 255_{10}
 00000000 00_{16} 000_{10}
 $\quad \quad \quad 0$ 0_{16} 0_{10}
 $[1] 11111110$ $1FE_{16}$ 510_{10}

NOT QUITE THE RESULTS ONE MIGHT HAVE WISHED FOR! (SUPERFICIALLY) THIS OCCURS BECAUSE OF THE HARDWARE IMPLEMENTATION OF SUBTRACTION A SUBTRACTION, (P-Q), IS REGARDED BY THE MICRO-PROCESSOR AS THE EQUIVALENT (P+(-Q)), BECAUSE THERE IS A SIMPLE WAY TO GENERATE THE NEGATIVE OF A NUMBER. THE 'ONES-COMPLEMENT' OF A BINARY NUMBER IS SIMPLY GENERATED BY EXCHANGING '0's & '1's:

'1's 00001100_2 $0C_{16}$ 12_{10}
COMPLEMENT 11110011_2 $F3_{16}$ 243_{10}
IF THIS ONE'S-COMPLEMENT IS TO BE THE NEGATIVE OF A NUMBER, WE SHOULD GET 0 ON ADDITION:

GLOSSARY

- ACCUMULATOR: 8-BIT CENTRAL REGISTER IN THE MICROPROCESSOR. MOST INFORMATION HAS TO GO THROUGH IT.
- ADDRESS: 16 BIT POINTER TO A MEMORY LOCATION. THE 6502 MICRO-PROCESSOR CAN ADDRESS 65, 536 SUCH LOCATIONS (WHICH IS 2^{16}).
- ARITHMETIC LOGIC UNIT (A.L.U.): A SECTION OF THE MICROPROCESSOR WHICH CARRIES OUT ARITHMETIC (ADDITION, SUBTRACTION, INCREMENT, DECREMENT & COMPARE) AND LOGIC ("AND", "EOR", "OR", & BIT SHIFTS) MANIPULATIONS. THIS IS THE ONLY PART OF THE MICROPROCESSOR WHICH ALTERS DATA.
- COMMAND: THE MONITOR FUNCTIONS M,G,P,R,L,S, \uparrow , \downarrow .
- DATA: INFORMATION FOR THE PROCESSOR THAT DOES NOT HAVE TO BE TRANSLATED. e.g. "AD" AS DATA ACTUALLY MEANS $10 \times 16 + 13 \times 1 = 173_{10}$ WHEREAS THE INSTRUCTION "AD" GETS TRANSLATED INTO THE OPERATION "LOAD ACCUMULATOR ABSOLUTE".
- EPROM: ERASABLE PROGRAMMABLE READ ONLY MEMORY. THIS TYPE OF MEMORY IS LIKE A PROM, BUT CAN AGAIN BE ERASED BY EXPOSING THE CHIP TO ULTRAVIOLET LIGHT.
- FLAGS: ONE BIT INTERNAL REGISTERS. ALL SEVEN FLAGS CAN ALSO BE TREATED AS SEPARATE BITS OF THE P REGISTER (PROCESSOR STATUS).
- INDEX REGISTER: A REGISTER WHICH CAN BE USED TO MODIFY AN ADDRESS (USED IN REFERRING TO DATA) BY BEING ADDED TO IT, THUS ACCESSING A CERTAIN ELEMENT OF A TABLE. THE 6502 HAS TWO INDEX REGISTERS CALLED X & Y.
- INSTRUCTION: A FUNCTION OF THE MICROPROCESSOR LIKE LOAD AND STORE.
- I/O: INPUT/OUTPUT. THIS CHIP ALLOWS YOU TO COMMUNICATE WITH THE OUTSIDE WORLD. IN THE ACORN THE I/O CHIP HAS 16 PROGRAMMABLE LINES WHICH CAN EITHER BE OUTPUTS OR INPUTS. IT ALSO HAS 128 BYTES OF RAM.
- IRQ: INTERRUPT REQUEST. IF FLAG I (INTERRUPT DISABLE) IS CLEAR AND A REQUEST IS MADE THE PROCESSOR WILL ATTEND TO IT AFTER SETTING FLAG I AND STORING THE PROGRAM COUNTER AND STATUS REGISTER.
- JUMP: THE PROGRAM COUNTER IS LOADED WITH A NEW ADDRESS. THE EXECUTION OF THE PROGRAM, WHICH IS NORMALLY USING CONSECUTIVE ADDRESSES, CONTINUES (JUMPS) AT THIS NEW ADDRESS.
- LOAD: TRANSFERS THE DATA OF A MEMORY LOCATION TO AN INTERNAL REGISTER.
- MNEMONIC: SUGGESTIVE ABBREVIATION OF AN INSTRUCTION e.g. THE INSTRUCTION "LOAD ACCUMULATOR ABSOLUTE" HAS THE MNEMONIC "LDA".
- NMI: NON MASKABLE INTERRUPT WHEN THE NON MASKABLE INTERRUPT IS ACTIVATED THE PROCESSOR WILL SET FLAG I, STORE AWAY ITS PROGRAM COUNTER AND STATUS REGISTER AND THEN IMMEDIATELY ATTEND TO THE INTERRUPT. THERE IS NO WAY OF PREVENTING THIS INTERRUPT. IT HAS PRIORITY OVER IRQ.
- OPCODE: HEXADECIMAL REPRESENTATION OF AN INSTRUCTION. e.g. THE INSTRUCTION "LOAD ACCUMULATOR ABSOLUTE" HAS THE MNEMONIC "LDA" AND THE OPCODE "AD".

0010	D,R4	BASE ADDRESS OF THE EIGHT DISPLAYED MEMORY LOCATIONS, REGISTER 4: TEMPORARILY PCH AFTER BREAK.
0011	R5	REGISTER 5: TEMPORARILY PCL AFTER BREAK
0012	R6	REGISTER 6: TEMPORARILY 01 AFTER BREAK
0013	R7	REGISTER 7: TEMPORARILY S AFTER BREAK.
0014-0017		LAST 4 DISPLAYED MEMORY LOCATIONS.
0018	P	SINGLE LEVEL OF STORAGE FOR PREVIOUS DATA AT BREAK POINTS.
0019	COL	COLUMN OF KEY CURRENTLY BEING PROCESSED
001A	TX,TY	TEMPORARY STORAGE FOR X (IN DISPLAY) OR Y (VARIOUS PLACES).
001C,001D	USERNMI	ADDRESS OF USER'S NMI PROGRAM
001E,001F	USERIRQ	ADDRESS OF USER'S IRQ PROGRAM
001B	RECAL	CONTAINS PC RECALCULATION FACTOR FOR BREAK
FE00	QUAD	DISPLAY X-3,X-2,X-1,X ON THE DISPLAY; THEN ↓
FE0C	DISPLAY	STROBE KEYBOARD, MULTIPLEX DISPLAY, RETURN WITH KEY INFORMATION
FE5E	MHEXTD	DISPLAY A MEMORY BYTE ON RIGHT OF DISPLAY
FE60	RDHEXTD	DISPLAY A ON RIGHT OF DISPLAY
FE64	QHEXTD1	DISPLAY X & X+1 ON DISPLAYS 1,2,3 & 4
FE66	QHEXTD2	DISPLAY X & X+1 ON DISPLAYS Y-2,Y-1, Y & Y+1
FE6F	DHEXTD	DISPLAY A ON DISPLAYS Y & Y+1
FE7A	HEXTD	DISPLAY BOTTOM 4 BITS OF A ON DISPLAY Y
FE88	QDATFET	FETCH AN ADDRESS INTO LOCATIONS X & X+1
FEA0	COM16	INCREMENT & COMPARE TWO 16 BIT NOS X+6,X+7 & X+8,X+9
FEA6	NOINC	COMPARE X+6,X+7 & X+8,X+9 FOR EQUALITY
FEB1	PUTBYTE	A TO TAPE, DO 1 START & 1 STOP BITS, NO PARITY
FECD	WAIT	WAIT FOR CASSETTE TIMING
FED0	½ WAIT	HALF THE WAIT
FEDD	GETBYTE	TAPE TO A, WAIT FOR START BIT, CENTRE TIMING
FEF3	RESET	ENTRY TO MONITOR
FF04	RESTART	RE-ENTRY TO RUNNING MONITOR
FFB3	BREAK	ENTRY TO MONITOR FROM BRK INSTRUCTION, DISPLAY CPU
FFEA	FONT	SEVEN SEGMENT PICTURES OF THE HEX DIGITS
001B	RECAL	CONTAINS PC RECALCULATION FACTOR FOR BREAK

$$\begin{array}{r} 00001100_2 \\ 11110011_2 + \end{array} \quad \begin{array}{r} 0C_{16} \\ F3_{16} + \end{array} \quad \begin{array}{r} 12_{10} \\ 243_{10} + \end{array}$$

$$\begin{array}{r} 11111111_2 \\ FF_{16} \end{array} \quad \begin{array}{r} 255_{10} \end{array}$$

WHICH DOESN'T HAPPEN UNTIL WE ADD AN EXTRA 1:

$$\begin{array}{r} 00001100_2 \\ 11110011_2 \\ \hline 1_2 + \\ 1 \ 00000000_2 \end{array} \quad \begin{array}{r} 0C_{16} \\ F3_{16} \\ \hline 1_{16} + \\ 100_{16} \end{array} \quad \begin{array}{r} 12_{10} \\ 243_{10} \\ \hline 1_{10} \\ 256_{10} \end{array} \quad 16$$

AND THEN TREAT THE OUTPUT CARRY AS INDICATING THE ABSENCE OF A BORROW FROM THE HIGHER ORDERS.

THE NUMBER (ONE'S-COMPLEMENT + 1) IS CALLED THE TWO'S-COMPLEMENT OF A NUMBER:

BINARY	HEXADECIMAL	DECIMAL
00000001 ₂	01 ₁₆	+1 ₁₀
00000000 ₂	00 ₁₆	+0 ₁₀ or -0 ₁₀
11111111 ₂	FF ₁₆	-1 ₁₀
11111110 ₂	FE ₁₆	-2 ₁₀
:	:	:
11110100 ₂	F4 ₁₆	-12 ₁₀
10000000 ₂	80 ₁₆	-128 ₁₀
01111111 ₂	7F ₁₆	+127 ₁₀

SO A BYTE CAN BE TREATED AS A 'SIGNED BINARY NUMBER' IN THE RANGE +127..... 0..... -128, OR AS A BINARY NUMBER IN THE RANGE 0.....+255. NOW THE SUBTRACTION ABOVE SHOULD BE CLEAR : INTERNALLY, THE MICRO-PROCESSOR ONE'S-COMPLEMENTS ONE OF THE NUMBERS AND THEN EXECUTES A NORMAL ADDITION WITH CARRY.

1.4 BINARY CODED DECIMAL (BCD) ARITHMETIC

99₁₆ LOOKS VERY LIKE 99₁₀. THEY BEHAVE THE SAME WAY AS THEY ARE MOVED AROUND AND UNDERGO LOGICAL OPERATIONS SINCE THEY ARE WRITTEN THE SAME WAY. THE BINARY REPRESENTATION OF 99₁₀ WOULD NORMALLY BE 01100011₂, AND OF 99₁₆ IT WOULD BE 10011001₂. WE NOW DEFINE THE BINARY CODED DECIMAL VERSION OF 99₁₀ AS BEING THE BINARY REPRESENTATION OF THE DECIMAL DIGITS IN THE ORIGINAL POSITIONAL NOTATION, MAKING THE DIFFERENCE BETWEEN THE BINARY REPRESENTATIONS OF 99₁₆ & 99₁₀ A MATTER OF SUPSCRIPTS:

$$\begin{array}{l} 99_{16} = 10011001_2 \\ 99_{10} = 10011001 \quad \text{B.C.D.} \end{array}$$

THE B.C.D. AND BINARY NUMBERS DIFFER IN HANDLING ONLY IN ARITHMETIC:

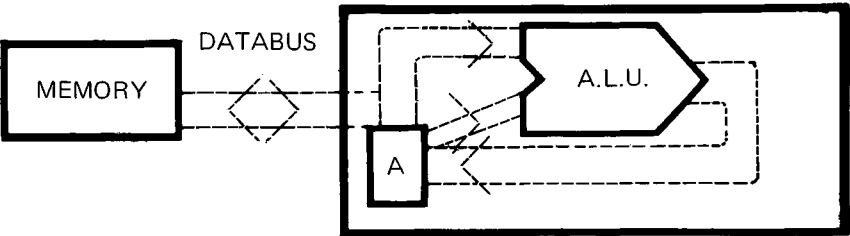
$$\begin{array}{r} 79_{16} \\ 22_{16} \\ 9B_{16} \end{array} + \quad \text{BUT} \quad \begin{array}{r} 79_{10} \\ 22_{10} \\ 101_{10} \end{array} +$$

THE MICROPROCESSOR CAN BE 'TOLD' WHICH TYPE OF ARITHMETIC TO CARRY OUT, BY SETTING (PUTTING A ONE INTO) OR CLEARING (PUTTING A ZERO INTO) AN INTERNAL BIT, THE 'DECIMAL MODE' FLAG.

CHAPTER 2: WELCOME TO THE MACHINE

2.1 HOW ACORN'S MICROPROCESSOR WORKS

TO CARRY OUT THE ABOVE OPERATIONS THE MICROPROCESSOR HAS AN INTERNAL ARITHMETIC LOGIC UNIT (A.L.U.) WHOSE OUTPUT IS SENT TO AN INTERNAL REGISTER OF ONE BYTE LENGTH CALLED THE ACCUMULATOR 'A', THIS REGISTER ALSO ACTS AS ONE OF THE OPERANDS; THE OTHER BEING DRAWN FROM THE MEMORY EXTERNAL TO THE μ PROCESSOR, WHICH IS CONNECTED TO THE μ P BY 8 LINES CALLED THE DATABUS:



DATA CAN BE TRANSFERRED ALONG THE DATABUS IN EITHER DIRECTION, THIS DIRECTION IS CHOSEN BY THE, μ P AND INDICATED TO THE EXTERNAL UNITS BY A SINGLE 'R/W' LINE : WHEN HIGH, '1', THE μ P IS RECEIVING DATA FROM THE MEMORY, 'READING'; WHEN LOW, '0', THE μ P IS SENDING DATA TO THE MEMORY, 'WRITING'. ALL INFORMATION USED BY THE μ P TRAVELS ALONG THE DATABUS, INCLUDING THE INSTRUCTIONS. SO THAT THE μ P KNOWS WHERE ITS INSTRUCTIONS ARE IT HAS A TWO BYTE (16₁₀ BIT) REGISTER CALLED THE PROGRAM COUNTER, 'PC', WHICH POINTS AT THE INSTRUCTIONS BEING EXECUTED. THE MEMORY CAN BE VIEWED AS A BOOK OF 256 PAGES, THE PARTICULAR PAGE BEING DECIDED BY THE MOST SIGNIFICANT 8 BITS (BITS 15–8) OF THE 16 BIT ADDRESS, EACH PAGE CONTAINING 256 BYTES, THE PARTICULAR BYTE BEING DECIDED BY THE LEAST SIGNIFICANT 8 BITS (BITS 7–0) OF THE 16 BIT ADDRESS.



IN THE KIT, PAGES FE₁₆ & FF₁₆ ARE OCCUPIED BY A NON-ERASEABLE PROGRAM TO INTERFACE BETWEEN THE MICROPROCESSOR AND THE KEYBOARD & DISPLAY UNIT. TO START THE μ P IN THIS PROGRAM (AT THE CORRECT PLACE) THERE IS A RESET BUTTON WHICH INITIALIZES THE PROGRAM COUNTER. IN PAGE 00₁₆ THERE IS SOME ALTERABLE MEMORY, OF WHICH THE BOTTOM IF₁₆ BYTES ARE GIVEN SPECIAL USES BY THE FE₁₆ & FF₁₆ MONITOR PROGRAM, SO, UNLESS PRESSED FOR SPACE, IT'S BEST TO STAY OUT OF THEM.

APPENDIX C HEXADECIMAL TO DECIMAL

1st DIGIT	2nd DIGIT															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

HEX	DEC
100	256
200	512
400	1024
800	2048
1000	4096
2000	8192
4000	16384
8000	32768
10000	65536

APPENDIX D ACORN MONITOR ADDRESS INFORMATION

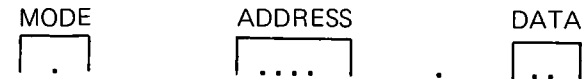
ADDRESS	LABEL	COMMENT
0000,0001	MAP	LOW AND HIGH BYTES OF THE M ADDRESS
0002,0003	GAP	LOW AND HIGH BYTES OF THE GO ADDRESS
0004,0005	PAP	LOW AND HIGH BYTES OF THE BREAKPOINT ADDRESS
0006,0007	FAP	LOW AND HIGH BYTES OF THE TAPE FROM ADDRESS
0008,0009	TAP	LOW AND HIGH BYTES OF THE TAPE TO ADDRESS
000A	R0	REGISTER 0: CONTAINS A AFTER BREAK.
000B	R1	REGISTER 1: CONTAINS X AFTER BREAK.
000C	R2	REGISTER 2: CONTAINS Y AFTER BREAK.
000D	R3, KEY	REGISTER 3: TEMPORARILY P AFTER BREAK, CONTAINS LAST PRESSED KEY FOR DISPLAY
000E	REPEAT	MSB=1 SETS REPEATEDLY SCANNED DISPLAY, OTHERWISE SINGLE SCAN.
000F	EXEC	EXECUTION STATUS OF THE KEY PROCESSING ROUTINE

1ST DIGIT	2ND DIGIT															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK (I,X)	ORD (I,X)				ORA ZERO	ASL ZERO		PHP	ORA IMMED	ASLA			ORA ABSOLUTE	ASL ABSOLUTE	
1	BPL (I),Y	ORD (I),Y				ORD Z,X	ASL Z,X		CLC	ORA A,Y				ORA A,X	ASL A,X	
2		AND (I,X)			BIT ZERO	AND ZERO	ROL ZERO		PLP	AND IMMED	ROLA		BIT ABSOLUTE	AND ABSOLUTE	ROL ABSOLUTE	
3	BMI (I),Y	AND (I),Y				AND Z,X	ROL Z,X			AND A,Y				AND A,X	ROL A,X	
4	RTI (I,X)	EOR (I,X)				EOR ZERO	LSR ZERO		PHA	EOR IMMED			JMP ABSOLUTE	EOR ABSOLUTE	LSR ABSOLUTE	
5	BVC (I),Y	EOR (I),Y				EOR Z,X	LSR Z,X		CLI	EOR A,Y	LSRA			EOR A,X	LSR A,X	
6	RTS (I,X)	ADC (I,X)				ADC ZERO	FOR ZERO		PLA	ADC IMMED	RORA		JMP INDIRECT	ADC ABS	ROR ABSOLUTE	
7	BVS (I),Y	ADC (I),Y				ADC Z,X	ROR Z,X			ADC A,Y				ADC A,X	ROR A,X	
8		STA (I,X)			STY ZERO	STA ZERO	STX ZERO		DEY	STA A,Y	TXA		STY ABSOLUTE	STA ABSOLUTE	STX ABSOLUTE	
9	BCC (I),Y	STA (I),Y			STY Z,X	STA Z,X	STX Z,X		TYA		TXS			STA A,X		
A	LDY IMMED (I,X)	LDA (I,X)		LDX IMMED	LDY ZERO	LDA ZERO	LDX ZERO		TAY	LDA IMMED	TAX		LDY ABSOLUTE	LDA ABSOLUTE	LDX ABSOLUTE	
B	BCS (I),Y	LDA (I),Y			LDY Z,X	LDA Z,X	LDX Z,X		CLV	LDA A,Y	TSX		LDY A,X	LDA A,X	LDX A,Y	
C	CPY IMMED (I,X)	CMP (I,X)			CPY ZERO	CMP ZERO	DEC ZERO		INY	CMP IMMED	DEX		CPY ABSOLUTE	CMP ABSOLUTE	DEC ABSOLUTE	
D	BNE (I),Y	CMP (I),Y				CMP Z,X	DEC Z,X		CLD	CMP A,Y				CMP A,X	DEC A,X	
E	CPX IMMED (I,X)	SBC (I,X)			CPX ZERO	SBC ZERO	INC ZERO		INX	SBC IMMED	NOP		CPX ABSOLUTE	SBC ABSOLUTE	INC ABSOLUTE	
F	BEQ (I),Y	SBC (I),Y				SBC Z,X	INC Z,X			SBC A,Y				SBC A,X	INC A,X	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

DISASSEMBLY CHART

2.2 THE MONITOR COMMANDS M,↑,↓

THE FIRST FEATURE OF THE MONITOR IS THE MEMORY INSPECT & MODIFY CONTROL SWITCH ON, AND PRESS THE RESET BUTTON:



THEN PRESS THE MODIFY KEY, M. THIS GETS YOU INTO THE MEMORY INSPECTION AND MODIFY MODE. THE MODE INDICATOR SHOWS 'A' FOR ALTER. THIS FIRST PHASE OF 'A' ALLOWS YOU TO CHOOSE ANY ADDRESS IN MEMORY.

A. XXXX .

APPEARS ON THE DISPLAY, WHERE X REPRESENTS ANY OF THE 16 HEX CHARACTERS SIGNIFYING THE ADDRESS. NOW PRESS THE KEYS F, E, 0, 0 (IF YOU MAKE A MISTAKE, E.G. PRESSED F, D, JUST START OFF FROM THE F AGAIN). AS EACH KEY IS PRESSED THE INFORMATION ON THE DISPLAY SHIFTS TO THE LEFT:

A. XXXF .
A. XXFE .
A. XFE0 .
A. FE00 .

AND SO YOU END UP WITH FE00 ON THE DISPLAY. PRESS ANY OF THE EIGHT COMMAND KEYS (IT DOES NOT MATTER WHICH) AND YOU CAN INSPECT THE CONTENTS OF THIS MEMORY ADDRESS. THIS IS PHASE TWO OF MODE 'A' AND ALLOWS YOU TO INSPECT AND ALTER THE DATA OF THE MEMORY ADDRESS CHOSEN IN PHASE ONE.

A. FE00 . A0

THIS IS THE INFORMATION STORED AT THE VERY BEGINNING OF THE MONITOR. IF YOU PRESS THE ↑ KEY

A. FE01 . 06

UP WE GO. NATURALLY THE ↓ KEY BRINGS BACK

A. FE00 . A0

AND EITHER KEY MAY BE USED ANY NUMBER OF TIMES IN SUCCESSION. NOW, IF, WITHOUT TURNING OFF, YOU PRESS RESET

AND THEN M

A. FE00 .

THE SYSTEM HAS REMEMBERED THE ADDRESS YOU WERE USING (WHICH DOESN'T HAVE TO BE FE00) TO INSPECT MEMORY NOW ENTER THE ADDRESS 0030 AND TERMINATE WITH ANY COMMAND KEY

A. 0030 . XX

0030 IS AN ADDRESS IN THE ALTERABLE SECTION OF THE MEMORY. PRESSING DIGIT KEYS NOW WILL CAUSE THE INFORMATION IN 0030 TO CHANGE (WHAT HAPPENS AT FE00?? TRY IT! YOU CANNOT WRITE INTO THE MONITOR PROM, (i.e. THE PROGRAMMABLE READ ONLY MEMORY). PRESS 0, 1.

A. 0030 . 01

PRESS 2,3

A. 0030 . 23

AS BEFORE INFORMATION IS SHIFTED IN UNTIL TERMINATED BY ANY COMMAND KEY. BUT, UNLIKE THE ADDRESS FETCHING PHASE, THE COMMAND KEY WILL BE EXECUTED. USEFUL TERMINATORS ARE THE M, ↑ & ↓ KEYS. PRESS ↑.

A. 0031 . XX

PRESS 4,5

A. 0031 . 45

PRESS ↓

A. 0030 . 23

& ↑ AGAIN

A 0031 . 45

YOU CAN GO UP AND DOWN INSPECTING & MODIFYING THE MEMORY CONTENTS IF THERE IS NO ALTERABLE MEMORY (E.G. A PROM) AT A PARTICULAR ADDRESS, THE INFORMATION WILL NOT CHANGE. TO CLOSE THIS SECTION WE'LL MAKE THE MONITOR DO A LITTLE TRICK. M,0,0,0,E, k (k ≡ ANY COMMAND KEY)

IV REGISTERS OTHER THAN ACCUMULATOR

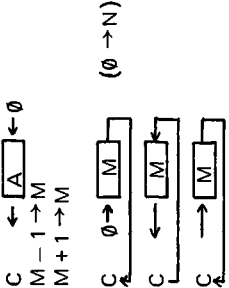
MNEMONIC	VERBAL	FLAGS	ADDRESSING MODE					A X A,Y INDIRECT					BYTES TIME
			2	2	2	2	3	3	3	3	3	5	
BIT			—	—	—	—	2C	—	—	—	—	—	A M → Z, M ₆ → V, M ₇ → N
CPX	LOGICAL AND WITH MASK AND TEST BITS	NZV	E0	E4	—	—	EC	—	—	—	—	—	X ← M
CPY	COMPARE X	NZC	C0	C4	—	—	CC	—	—	—	—	—	Y ← M
JMP	COMPARE Y	NZC	—	—	—	—	4C	—	—	—	—	—	M → PCL M + 1 → PCH
LDX	JUMP	—	A2	A6	—	—	B6	—	—	—	—	—	M → X
LDY	LOAD X REGISTER	NZ	A0	A4	—	—	AE	—	—	—	—	—	M → Y
STX	LOAD Y REGISTER	NZ	—	86	—	—	96	—	—	—	—	—	X → M
STY	STORE X REGISTER	—	—	84	—	—	8E	—	—	—	—	—	Y → M
	STORE Y REGISTER	—	—	84	—	—	8C	—	—	—	—	—	

V READ – MODIFY – WRITE

MNEMONIC	VERBAL	FLAGS	ADDRESSING MODE					A,X				
			2	2	2	2	3	3	3	3	3	7
ASL	ARITHMETIC SHIFT LEFT	NZC	06	16	0E	—	1E	—	—	—	—	—
DEC	BINARY DECREMENT	NZ	C6	D6	CE	—	DE	—	—	—	—	—
INC	BINARY INCREMENT	NZ	E6	F6	EE	—	FE	—	—	—	—	—
LSR	LOGICAL SHIFT RIGHT	NZC	46	56	4E	—	5E	—	—	—	—	—
ROL	ROTATE LEFT	NZC	26	36	2E	—	3E	—	—	—	—	—
ROR	ROTATE RIGHT	NZC	66	76	6E	—	7E	—	—	—	—	—

+ : EXTRA CYCLE IF OPERATION INVOLVES PAGE CROSSING
+↑: + 0 IF NO BRANCH; + 1 IF BRANCH; + 2 IF BRANCH & CROSS PAGE

BYTES TIME



$$-8 = F_8 \Rightarrow \begin{matrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 & 40 & 44 & 48 & 52 & 56 & 60 & 64 & 68 & 72 & 76 & 80 & 84 & 88 & 92 & 96 & 100 \end{matrix}$$

1
2
3
4
5

2.3 AT LAST, A PROGRAM

A PROGRAM IS THE NAME FOR A SET OF STORED COMMANDS THAT THE MICROPROCESSOR WILL EXECUTE. THESE ARE STORED IN BINARY, SINCE THAT'S ALL THAT ANYTHING CAN BE STORED IN, (ENTERED BY YOU IN HEX) AND ARE INDISTINGUISHABLE FROM ANYTHING ELSE. IF IT GETS THE CHANCE THE μP (MICROPROCESSOR) WILL BUSY ITSELF TREATING INFORMATION WHICH YOU MEANT AS DATA AS A PROGRAM. IT PROBABLY WON'T BE DOING ANYTHING INTELLIGENT AND WILL HAVE TO BE SUMMONED BACK WITH THE RESET KEY. SOME SORT OF TRANSLATION BETWEEN THE STORED BINARY/HEX AND YOU IS NEEDED. 10101101_2 MEANS A GREAT DEAL TO THE μP BUT LITTLE TO YOU. IT ACTUALLY MEANS "LOAD THE ACCUMULATOR WITH THE CONTENTS OF THE MEMORY ADDRESS DEFINED BY THE FOLLOWING TWO BYTES, OF WHICH THE FIRST IS THE LEAST SIGNIFICANT ADDRESS". THIS IS A LITTLE LONG FOR WRITING STRAIGHT INTO A PROGRAM AND IS USUALLY ABBREVIATED TO LDA ABS, OR JUST LDA. ABSOLUTE MEANS ANYWHERE IN THE 64K. THE 6502 CAN ADDRESS 64K OF MEMORY WHICH IS DIVIDED INTO PAGES 256 BYTES LONG THE FIRST PAGE IS CALLED ZERO PAGE. LOCATIONS IN ZERO PAGE CAN BE ADDRESSED BY JUST ONE BYTE. THERE ARE SPECIAL INSTRUCTIONS TO DO THIS. AT THE END OF THE MANUAL THERE IS A LIST OF ALL THESE MNEMONICS WITH THEIR HEX EQUIVALENTS IN APPENDIX B. SO IF WE WROTE THE PROGRAM IN MNEMONICS IT WOULD LOOK LIKE.

AND WE WOULD TRANSLATE IT FOR THE μP AS THE THREE BYTES

WHICH WOULD CAUSE THE μP TO PUT A0 (THE DATA STORED IN FE00) IN ITS ACCUMULATOR (REMEMBER USING THE MONITOR TO LOOK AT FE00?). THE TRANSLATION PROCESS IS CALLED ASSEMBLING AND COMPUTER PROGRAMS WHICH DO IT ARE CALLED ASSEMBLERS. A RESIDENT ASSEMBLER IS ONE THAT RUNS (OPERATES) ON THE SAME MACHINE THAT IT ASSEMBLES FOR; A CROSS ASSEMBLER RUNS ON A DIFFERENT MACHINE. THE MNEMONICS LDA, STA etc ARE OFTEN CALLED ASSEMBLY LANGUAGE, THE GENERATED BINARY IS CALLED MACHINE CODE. WE CAN LOAD THE ACCUMULATOR IN TEN OTHER WAYS; HERE ARE TWO OF THEM.

INSTRUCTION
LENGTH
IN

BYTES	TYPE	HEX	MNEMONIC	EXECUTION TIME μ S	BRIEF EXPLANATION
2	1	A9	LDA #	2	PUT THE NEXT BYTE IN ACCUMULATOR. "LOAD IMMEDIATE".
2	2	A5	LDA Z	3	SHORTENED FORM OF LOAD ABS 00XX 'LOAD ZERO PAGE'.
3	3	AD	LDA	4	LOAD A ABSOLUTE.

THE FIRST OF THESE INSTRUCTIONS IS VERY IMPORTANT. IF WE KNOW THAT WE WANT A0 IN THE ACCUMULATOR THEN IT IS WASTEFUL TO FIND A MEMORY LOCATION WHICH HAPPENS TO CONTAIN IT, SINCE TWO BYTES ARE NEEDED (GENERALLY) TO SPECIFY WHERE IT IS AND SO WE IMPLY, BY THE IMMEDIATE INSTRUCTION, WHERE IT IS & ACTUALLY ENTER IT IN THE PROGRAM. THERE ARE COMPLEMENTARY STORE ACCUMULATOR 'STA' INSTRUCTIONS TO LDA Z AND LDA.

BYTES	TYPE	HEX	MNEMONIC	TIME μ S	
2	2	85	STA Z	2	STORE A ZERO PAGE (IN THE FIRST 256 BYTES)
3	3	8D	STA	3	STORE A ABSOLUTE (ANYWHERE IN MEMORY)

WE CAN ALSO LOAD THE PROGRAM COUNTER. THE PROGRAM COUNTER IS AN INTERNAL REGISTER THAT POINTS TO THE NEXT LINE OF THE PROGRAM. THE MNEMONIC FOR THIS IS NOT LDPC BECAUSE WHEN THE P.C. IS LOADED WITH A NEW VALUE IT GIVES THE MICROPROCESSOR A DIFFERENT PLACE TO LOOK FOR INSTRUCTIONS: THE PROGRAM JUMPS. SO 'LOAD P.C. WITH NEXT TWO BYTES' (LDPC) IS JMP, THIS IS REFERRED TO AS JUMP ABSOLUTE SINCE THE PROGRAM JUMPS TO A NEW ABSOLUTE ADDRESS. SO IF WE ARE NOT IN THE MONITOR AND WANT TO BE, JMP FF04 WILL ENTER THE MONITOR. NOW WHAT HAPPENS IF THE FOLLOWING PROGRAM IS RUN?

```

LDA    FE00
STA    Z    20
JMP    FF04

```

THE FIRST INSTRUCTION GETS THE CONTENTS OF FE00, AND PUTS IT IN THE ACCUMULATOR. THE SECOND STORES THE ACCUMULATOR IN LOCATION 0020. THE FIRST TWO 0'S REFER TO ZERO PAGE AND ARE ASSUMED BY THE PROCESSOR IN THE ZERO PAGE MODE. THE THIRD GETS BACK TO THE MONITOR, SO THAT YOU CAN INSPECT LOCATION 20. THIS READS AS.

```

0030    AD (OPCODE)    LDA FE00
0031    00 (DATA)
0032    FE (DATA)
0033    85 (OPCODE)    STA Z 20
0034    20 (DATA)
0035    4C (OPCODE)    JMP FF04
0036    04 (DATA)
0037    FF (DATA)

```

II RELATIVE: RELATIVE ADDRESSING MODE

2 BYTES 2+1 CYCLES

MNEMONIC	VERBAL		
BCC	BRANCH IF CARRY CLEAR	90	BRANCH IF C = 0
BCS	BRANCH IF CARRY SET	B0	C = 1
BEQ	BRANCH IF EQUAL (TO ZERO)	F0	Z = 1
BMI	BRANCH IF MINUS	30	N = 1
BNE	BRANCH IF NOT EQUAL	D0	Z = 0
BPL	BRANCH IF PLUS	10	N = 0
BVC	BRANCH IF OVERFLOW CLEAR	50	V = 0
BVS	BRANCH IF OVERFLOW SET	70	V = 1

APPENDIX B INSTRUCTION SET

I ACCUMULATOR REFERENCE: ACCUMULATOR, OPERATION, MEMORY → ACCUMULATOR

MNEMONIC	VERBAL	ADDRESSING MODE		IMMED	ZERO	Z,X	ABSOLUTE	A,X	A,Y	(I,X)	(I),Y	BYTES CYCLES = SPEED μ S
		FLAGS IN P AFFECTED	2	2	3	3	3	3	3	2	2	
		2	3	4	4	4	4	4+	4+	6	5+	
ADC AND CMP EOR LDA ORA SBC	ADD WITH CARRY LOGICAL AND LOGICAL COMPARE LOGICAL EXCLUSIVE OR LOAD ACCUMULATOR LOGICAL OR SUBTRACT WITH BORROW/CARRY	NZCV	69	65	75	6D	7D	79	61	71	A + M + C \rightarrow A	
		NZ	29	25	35	2D	3D	39	21	31	A \rightarrow M \rightarrow A	
		NZC	C9	C5	D5	CD	DD	D9	C1	D1	A - M	
		NZ	49	45	55	4D	5D	59	41	51	A VM \rightarrow A	
		NZ	A9	A5	B5	AD	BD	B9	A1	B1	M \rightarrow A	
		NZ	09	05	15	0D	1D	19	01	11	A VM \rightarrow A	
		NZCV	E9	E5	F5	ED	FD	F9	E1	F1	A - M + C \rightarrow A	
			-	85	95	8D	9D	99	81	91	A \rightarrow M	
STA	STORE ACCUMULATOR											

BYTES
CYCLES =
SPEED μ S

A + M + C → A
A → M → A
A - M

A V M → A
M → A
A V M → A

A - M + C → A
A → M

THE ADDRESS 0030 IS THE STARTING ADDRESS OF THE PROGRAM. THIS PARTICULAR PROGRAM WILL WORK WITH ANY STARTING ADDRESS – IT IS SAID TO BE 'POSITION INDEPENDENT' OR 'RELOCATABLE' – BUT OTHER PROGRAMS MAY NOT. IF YOU ARE NEW TO THE GAME, IT WILL BE EASIER IF YOU ENTER PROGRAMS AT THE STARTING ADDRESS SHOWN IN THE MANUAL.

2.3.2 ENTERING A PROGRAM, THE GO COMMAND

TO ENTER THIS PROGRAM, WE'LL GO THROUGH IT STEP BY STEP.

- ENTER THE STARTING ADDRESS: PRESS M,0,0,3,0, k
- ENTER A BYTE OF DATA A,D
- USE THE ↑ KEY TO TERMINATE DATA ENTRY AND STEP UP
– CONTINUE WITH 0,0,↑,F,E,↑,8,5,↑,2,0,↑,4,C,↑,0,4,↑,F,F
- CHECK THAT THE PROGRAM IS ENTERED CORRECTLY BY, E.G, USING ↓
TO GO BACK DOWN THROUGH IT.
– REMEMBER THAT MISTAKES AT KEY ENTRY (E.G. PRESSED 8,6) MAY BE CORRECTED BY CONTINUING (PRESS 8,5) –
NOW THAT THE PROGRAM IS LOADED PRESS ONLY ONCE THE 'GO' (G) KEY

K. XXXX .

APPEARS THE K (F.) REMINDS YOU OF TWO THINGS: 1 THIS IS A DIFFERENT STORED ADDRESS TO THE A. ADDRESS. 2 YOU CAN'T GO BACK! (UNLESS YOU EITHER PRESS RESET OR ENTER ADDRESS FF04, THE MONITOR ENTRY ADDRESS, AND GO) THE NEXT COMMAND KEY YOU PRESS WILL CAUSE THE μ P TO DO A KAMI-KAZE DIVE TO THE ADDRESS SHOWN, SO ITS AS WELL TO GET IT RIGHT!! ENTER 0,0,3,0

K. 0030 .

AND PRESS ANY COMMAND KEY. NOTHING HAPPENED? WELL IT DID, REALLY. IT JUST HAPPENED VERY QUICKLY:

PROGRAM	EXECUTION TIMES, μ S
LDA FE00	4
STA Z 20	3
JMP FF04	3
TOTAL 10 ₁₀ μ S	

IT TOOK TEN MILLIONTHS OF A SECOND TO HAPPEN. WE'RE NOW BACK IN THE MONITOR. PRESSING ANY DIGIT KEY WILL CAUSE THE (BY NOW) FAMILIAR DOTS TO REAPPEAR. PRESS M,0,0,2,0 k :

A. 0020 . A0

WHICH CHECKS THAT THE PROGRAM ACTUALLY DID WORK. YOU COULD CHANGE 0020 AND RUN THE PROGRAM AGAIN BY THE KEYS

F, F, G, G, M, M

WHICH SUCCESSIVELY PUT FF IN 0020, RUN THE PROGRAM AND RE-EXAMINE

LOCATION 0020. A LOT QUICKER FOR YOU THE SECOND TIME, WASN'T IT? THIS IS BECAUSE M & G REMEMBER WHAT THEY WERE POINTING AT. LET'S MAKE THE PROGRAM BETTER. AT THE MOMENT WE HAVE NO IDEA IF IT RAN, AND WE DON'T KNOW IF IT RAN CORRECTLY UNTIL WE LOOK AT 0020. IF THE PROGRAM WROTE OUT THE BYTE ON THE DISPLAY AS WELL AS STORING IT IN 0020, WE'D KNOW THAT IT HAD ALL HAPPENED. INSIDE THE ACORN MONITOR PROGRAM IS A SET OF INSTRUCTIONS TO WRITE A BYTE ONTO THE TWO RIGHT HAND DISPLAY DIGITS. THIS PROGRAM IS LOCATED AT FE60 AND EXPECTS THE BYTE TO BE DISPLAYED TO BE IN THE ACCUMULATOR, WHICH IT IS. THE PROGRAM DESTROYS THIS BYTE AS IT PUTS IT ONTO THE DISPLAY SO WE MUST PUT IT IN 0020 BEFORE USING THE PROGRAM.

2.3.3 INSTRUCTIONS JMP, JSR

IF WE SIMPLY WENT JMP FE60 THIS WOULD CORRECTLY EXECUTE THE PROGRAM BUT WE WOULD BE LEFT IN THE MIDDLE OF THE MONITOR SOMEWHERE SINCE THE PROGRAM DOES NOT HAVE AN ADDRESS TO JUMP BACK TO. WE CAN GIVE IT SUCH AN ADDRESS WITH THE INSTRUCTION JSR (OPCODE 20 HEX) THIS IS EXACTLY LIKE A JUMP BUT IT SAVES THE PROGRAM COUNTER BEFORE JUMPING. THEN THE SINGLE BYTE INSTRUCTION RTS (OPCODE 60 HEX) RESTORES THE PROGRAM COUNTER AND WE GET BACK AGAIN. JSR IS "JUMP TO SUBROUTINE" AND RTS IS "RETURN FROM SUBROUTINE". THE PROGRAM AT FE60 HAS AN RTS ATTACHED AT ITS END, AND SO CAN TRANSFER CONTROL BACK TO THE PROGRAM WHICH CALLED IT. OUR NEW PROGRAM IS 3 BYTES LONGER:

0030	AD	LDA FE00
0031	00	
0032	FE	
0033	85	STA Z 20
0034	20	
0035	20	JSR FE60
0036	60	
0037	FE	
0038	4C	JMP FF04
0039	04	
003A	FF	

AND WE WILL HAVE TO ENTER 6 BYTES FROM 0035 TO 003A WITH M,0,0,3,5, k , 2,0,↑,6,D,↑,F,E,↑,4,C,↑,0,4,↑,F,F. WE HAVEN'T CHANGED THE START OF THE PROGRAM SO G, G WILL RUN IT.

K. 0030 . A0

APPEARS MEANING THAT 0020 HAS AGAIN HAD A0 WRITTEN INTO IT. INSTEAD OF STORING THINGS IN 0020, LET'S USE ITS INFORMATION AS PART OF A LOGICAL OPERATION.

APPENDIX A
64 CHARACTER ASCII ON ACORN'S 7 SEGMENT DISPLAY

ASCII CODE	DISPLAY	CHARACTER	HEX	ASCII CODE	DISPLAY	CHARACTER	HEX
0	␣	@	5F	20			00
1	␣	A	77	21	!	!	86
2	␣	B	7C	22	"	"	22
3	␣	C	58	23	#	#	63
4	␣	D	5E	24	£	£	3B
5	␣	E	79	25	%	%	2D
6	␣	F	71	26	&	&	7B
7	␣	G	3D	27	'	'	02
8	␣	H	34	28	((B9
9	␣	I	05	29))	8F
A	␣	J	0D	2A	*	*	76
B	␣	K	75	2B	+	+	42
C	␣	L	38	2C	,	,	04
D	␣	M	37	2D	-	-	40
E	␣	N	54	2E	.	.	80
F	␣	O	5C	2F	/	/	52
10	␣	P	73	30	0	0	3F
11	␣	Q	67	31	1	1	06
12	␣	R	50	32	2	2	5B
13	␣	S	ED	33	3	3	4F
14	␣	T	78	34	4	4	66
15	␣	U	9C	35	5	5	6D
16	␣	V	1C	36	6	6	7D
17	␣	W	7E	37	7	7	07
18	␣	X	49	38	8	8	7F
19	␣	Y	6E	39	9	9	6F
1A	␣	Z	BD	3A	:	:	82
1B	␣	[39	3B	;	;	84
1C	␣	\	64	3C	<	<	46
1D	␣]	0F	3D	=	=	48
1E	␣	^	23	3E	>	>	70
1F	␣	_	08	3F	?	?	D3

PROGRAM. THE STRATEGY OF THE PROGRAM IS NOT OBVIOUS, AND IS LEFT AS AN EXERCISE TO THE READER. A SMALL PRIZE WILL REWARD THE SUBMISSION OF A SHORTER, FASTER PROGRAM; NOTE THAT WORKSPACE REQUIREMENTS CONTRIBUTE TO THE LENGTH!

8 QUEENS PROGRAM

```

0200 F8      MAIN    SED
0201 A2 20      LDX #20
0203 84 1F      STY COUNT    - CLEAR COUNT
0205 84 20      STY ROW      - CLEAR ROW OCCUPIED
0207 84 29      STY LEFT     - CLEAR LEFT DIAGONAL ATTACKS
0209 84 32      STY RIGHT    - CLEAR RIGHT DIAGONAL ATTACKS
020B 20 16 02   JSR TRY      - FIND THE NO OF WAYS
020E A5 1F      LDA COUNT
0210 20 60 FE   JSR RDHEXTD   - DISPLAY ANSWER
0213 4C 04 FF   JMP RESTART
0216 B5 00      TRY      LDAZX 00    - FINISHED YET?
0218 C9 FF      CMP #FF
021A D0 07      BNE CONTINUE
021C A5 1F      LDA COUNT    - FINISHED, SO INCREMENT COUNT
021E 69 00      ADC #00
0220 85 1F      STA COUNT
0222 60          RTS
0223 15 09      FINISH  ORAZX 09    - CURRENT LEFT
0225 15 12      ORAZX 12    - CURRENT RIGHT
0227 A8          LOOP      TAY
0228 49 FF      EOR #FF
022A F0 F6      BEQ FINISH    - NO CHANCE
022C 95 18      STAZX 1B     - CURRENT POSSIBLE PLACE
022E C8          INY
022F 98          TAY
0230 35 1B      ANDZX 1B
0232 A8          TAY
0233 15 00      CRAZX 00
0235 95 01      STAZX 01    - NEW ROW
0237 98          TYA
0238 15 09      ORAZX 09
023A 0A          ASLA
023B 95 0A      STAZX 0A    - NEW LEFT ATTACK
023D 98          TYA
023E 15 12      ORAZX 12
0240 4A          LSRA
0241 95 13      STAZX 13    - NEW RIGHT ATTACK
0243 E8          INX
0244 20 16 02   JSR TRY
0247 CA          DEX
0248 B5 01      LDAZX 01
024A 49 FF      EOR #FF
024C 35 1B      ANDZX 1B
024E 49 FF      EOR #FF
0250 4C 27 02   JMP LOOP
0253

```

2.3.4 THE LOGIC INSTRUCTIONS 'ORA', 'AND', 'EOR'.

IF WE PUT 60_{16} IN LOCATION 0020 (M,0,0,2,0, k, 6,0 : YOU SHOULD KNOW BY NOW) AND ALTER THE STA Z INSTRUCTION AT 0033 TO, SAY, ORA Z (OPCODE 05 HEX) (THE PROGRAM READS

```

LDA FE00
ORA Z 20
JSR FE60
JMP FF04)

```

WE HAVE A PROGRAM THAT DISPLAYS THE LOGICAL 'OR' BETWEEN THE CONTENTS OF $FE00$ (A0) AND 0020 , (60). THE HEX FOR ORA Z IS 05 AND IT CARRIES OUT A LOGICAL 'OR' BETWEEN THE ACCUMULATOR AND THE SPECIFIED LOCATION IN Z PAGE. M,0,0,3,3, k 0,5 IS THE MODIFICATION TO THE PROGRAM, THEN SINCE WE STILL START AT 0030 , G,G RUNS IT :

K. 0030 E 0

THE OPERATION WAS 'OR' :

```

A0      10100000
60 or   01100000 or
E0      11100000

```

TRY CHANGING 0020 TO 40 AND RUNNING THE PROGRAM AGAIN IS THE ANSWER WHAT YOU EXPECTED?

WE CAN CHANGE 0033 TO MAKE THE PROGRAM DO LOGICAL 'AND' OR 'EXCLUSIVE - OR'. THE MNEMONICS AND OPCODES ARE:

```

AND Z 2516 LOGICAL AND ACCUMULATOR AND Z PAGE
                MEMORY
EOR Z 4516 LOGICAL EXCLUSIVE-OR ACCUMULATOR AND
                Z PAGE MEMORY

```

AND THE PROGRAMS WOULD READ

```

LDA FE00      &   LDA FE00
AND Z 20      EOR Z 20
JSR FE60      JSR FE60
JMP FF04      JMP FF04

```

BY NOW YOU MUST BE GETTING TIRED OF THE A0 IN $FE00$ SO WE'LL CHANGE THE PROGRAM TO READ

```

LDA Z 21
EOR Z 20
JSR FE60
JMP FF04

```

THE SPACE TAKEN UP BY $LDA Z 21$ IS ONE BYTE LESS THAN THAT USED BY $LDA FE00$. WE COULD SIMPLY WRITE THE NEW TWO BYTES IN AT LOCATIONS 0031 & 0032 AND CHANGE THE GO ADDRESS TO 0031 . THIS IS VERY SIMPLE HERE SINCE THAT IS ALL WE HAVE TO DO. BUT IF THERE WERE MANY REFERENCES TO 0030 AS THE START OF THIS PROGRAM IT WOULD TAKE A LONG TIME TO FIND AND CHANGE THEM ALL, AND IF WE DIDN'T CHANGE THEM ALL SOMETHING WOULD GO WRONG. WE CAN'T MOVE THE REST OF THE PROGRAM DOWN ONE BYTE: SOMETHING MIGHT BE REFERRING TO IT. THE PROBLEM ARISES BECAUSE $LDA Z$ IS SHORTER THAN LDA . WE COULD SIMPLY USE LDA WITH A ZERO PAGE ADDRESS BUT THIS TAKES A WHOLE μs

LONGER THAN LDA Z! THE SOLUTION IS TO USE LDA Z AND TO INCORPORATE AN EXTRA BYTE IN 0030 AS PADDING. THIS MUST BE A SINGLE-BYTE INSTRUCTION, THAT DOES NOTHING TO AFFECT THE PROGRAM, AND ONE IS SPECIFICALLY PROVIDED

NOP EA "NO OPERATION"

THE PROGRAM READS

```
0030 EA      NOP
0031 A5 21   LDA Z 21
0033 45 20   EOR Z 20
0035 26 60 FE JSR FE60
0038 4C 04 FF JMP FF04
```

—NOTICE THE MORE COMPACT MODE OF WRITING IT DOWN. THIS IS MORE CONSISTENT WITH THE WAY MNEMONICS ARE WRITTEN. IT IS EXACTLY EQUIVALENT TO

```
0030 EA      NOP
0031 A5      LDA Z 21
0032 21
0033 45      EOR Z 20
0034 20
0035 20      JSR FE60
0036 60
0037 FE
0038 4C      JMP FF04
0039 04
003A FF
```

AND IT WILL BE USED THROUGHOUT THE REST OF THE MANUAL: THIS PROGRAM TAKES THE CONTENTS OF (WHICH MAY BE WRITTEN BY PUTTING BRACKETS AROUND THE PARTICULAR ADDRESS) 0020 & 0021 AND PRESENTS THEIR LOGICAL EXCLUSIVE — OR ON THE DISPLAY. APART FROM THEIR LOGICAL FUNCTIONS, THESE OPERATORS ARE OFTEN USED TO MANIPULATE SINGLE BITS. FOR INSTANCE ORA #01 WOULD SET BIT 0 OF THE ACCUMULATOR, AND #FE WOULD CLEAR IT AND EOR #01 WOULD COMPLEMENT IT, ALL WITHOUT AFFECTING ANY OTHER BITS IN THE ACCUMULATOR.

2.3.5 ARITHMETIC INSTRUCTIONS 'ADC', 'SEC', 'CLC'.

FROM LOGIC OPERATIONS WE PROGRESS AGAIN TO ARITHMETIC. LOOKING AT ORA Z, EOR Z, AND Z WOULD LEAD ONE TO ASSUME THE EXISTENCE OF ADD Z. WELL, THERE ISN'T ONE, THERE'S ONLY ADC Z.

BYTES: 2 ADC Z 65 "ADD WITH CARRY, ZERO PAGE"
 1 SEC 38 "SET CARRY FLAG"
 1 CLC 18 "CLEAR CARRY FLAG"

THIS IS MOST UNUSUAL AND A TRAP FOR UNWARY PROGRAMMERS, ESPECIALLY THOSE USED TO μ Ps WHICH POSSESS AN ADD INSTRUCTION: THE CARRY FLAG MUST BE CLEARED BEFORE AN ADC (OR IT MUST BE IN A KNOWN STATE E.G.

$\left\{ \begin{array}{l} \text{SEC} \\ \text{ADC \#00} \end{array} \right\} \equiv \left\{ \begin{array}{l} \text{CLC} \\ \text{ADC \#01 OR} \end{array} \right\}$

'UNEXPECTED' ANSWERS WILL APPEAR. WHEN THE μ P LEAVES THE MONITOR USING THE GO ROUTINE THE CARRY FLAG IS SET: FAILURE TO CLEAR IT BEFORE AN ADC RESULTS IN AN ANSWER 1 GREATER THAN EXPECTED.

3

THE METRONOME PRODUCES A PULSE AT THE TAPE OUTPUT PIN, PA6, WITH A REGULAR PERIOD. THE "UP" AND "DOWN" KEYS WILL INCREASE AND DECREASE THE PERIOD RESPECTIVELY. WITH SUITABLE ADDITIONAL CIRCUITRY THIS COULD DRIVE A LOUDSPEAKER OR A 'STROBE' LIGHT. IN FACT A SMALL SOUND CAN BE OBTAINED BY SIMPLY CONNECTING A LOUD-SPEAKER ACROSS THE TAPE OUTPUT AND EARTH PINS.

THE CONSTANTS USED AT PRESENT MEAN THAT THE PULSE IS OF 1/300 SEC. AND THE DELAY BETWEEN PULSES CAN BE VARIED FROM 1/20 SEC. TO ABOUT 13 SECS. YOU CAN DEFINE THE PERIOD BEFORE STARTING THE PROGRAM BY PUTTING THE REQUIRED VALUE INTO MEMORY LOCATION 0020. 20 WILL GIVE ABOUT 1 SEC BETWEEN PULSES, AND ANYTHING ELSE PROPORTIONATELY MORE OR LESS. ONCE THE PROGRAM IS RUNNING THE 'UP' AND 'DOWN' KEYS WILL INCREMENT AND DECREMENT THE PERIOD BY ABOUT 1/20 SEC EACH TIME THEY ARE PRESSED. THEY ALSO RESET THE CYCLE. THIS FACILITY COULD USEFULLY BE USED FOR FINE TUNING BUT WOULD BE TEDIOUS FOR LARGE CHANGES OF PERIOD.

METRONOME

ADDR	HEX	LABEL	INSTRUCTION	COMMENTS
	CODE			
0200	A9 1F		LDA #1F	
0202	85 0E		STA REPEAT	— SET DISPLAY TO SINGLE SCAN
0204	A9 40	PULSE	LDA #40	
0206	8D 22 0E		STA 1ADDR	— DEFINE PA6 AS OUTPUT
0209	8D 16 0E		STA SET PIA6	— USE INS8154 SET BIT MODE
020C	20 CD FE		JSR WAIT	— USE THE 300 BAND WAIT
020F	8D 06 0E		STA CLR PIA6	— USE IN58154 CLEAR BIT MODE
0212	A6 20		LDX# PERIOD	
0214	20 0C FE	DELZ	JSR DISPLAY	— LOOK AT KEYBOARD
0217	C9 16		CMP #16	— UP KEY?
0219	D0 04		BNE DOWN	— NO
021B	E6 20		INCZ PERIOD	— INCREASE PERIOD
021D	B0 E5		BCS PULSE	— CARRY WAS SET BY THE COMPARE: ALWAYS
021F	C9 17	DOWN	CMP #17	— DOWN KEY?
0221	D0 04		BNE DELI	— NO
0223	C6 20		DECZ PERIOD	— DECREASE PERIOD
0225	B0 DD		BCS PULSE	— CARRY WAS SET BY THE COMPARE: ALWAYS
0227	A0 0C	DELI	LDY #0C	— CYCLE TIME OF μ 1/30 SEC.
0229	20 CD FE	DELJ	JSR WAIT	
022C	88		DEY	
022D	10 FA		BPL DELJ	
022F	CA		DEX	
0230	D0 E2		BNE DEL2	
0232	F0 D0		BEQ PULSE	— END OF THIS PERIOD SO PULSE
0234				

4

THE EIGHT QUEENS PROBLEM IS TO FIND THE NUMBER OF WAYS IN WHICH EIGHT QUEENS MAY BE PLACED ON A CHESS BOARD WITHOUT ATTACKING EACH OTHER. THE PROGRAM FINDS 92 WAYS SINCE IT COUNTS ROTATIONS AND REFLECTIONS, ALL POSSIBLE POSITIONS ARE TRIED AS SOLUTIONS IN THIS HIGH SPEED RECURSIVE (I.E. IS DEFINED IN TERMS OF ITSELF)

COUNTER KEYBOARD

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
001D	20 0C	FE DISP	JSR DISPLAY	— START OF 001C
0020	90 0A		BCC CHANGE	— LOOK FOR KEY
				— CHECK IF CONTROL KEY CARRY SET IF SO
0022	C9 07		CMP # 07	
0024	F0 1F		BEQ DOWN	
0026	C9 06		CMP # 06	
0028	F0 11		BEQ UP	
002A	D0 F1		BNE DISP	
002C	C9 00	CHANGE	CMP # 00	
002E	85 19		STA COUNT	
0030	F0 ED	MORE	BEQ DISP	
0032	20 60 00		JSR INCR	INCREMENT NO OF TIME OF TEY
0035	C6 19		DEC COUNT	
0037	10 F7		BPL MORE	
0039	30 E2		BMI DISP	
003B	20 60 00	UP	JSR INCR	RAPID INCREMENT
003E	20 45 00		JSR ZOOM	
0041	D0 CF		BNE DISP	
0043	F0 F6		BEQ UP	
0045	20 69 00	DOWN	JSR DECR	RAPID INCREMENT
0048	20 4F 00		JSR ZOOM	
004B	D0 D9		BNE DISP	
004D	F0 F6		BEQ DOWN	
004F	A9 1F	ZOOM	LDA # 1F	
0051	85 0E		STA 0E	SET FOR ONE SCAN ONLY
0053	20 0C	FE	JSR DISPLAY	
0056	90 03		BCC STOP	CHECK IF KEY DEPRESSED CLEAR IF ONE IS
0058	A9 00		LDA # 00	
005A	60		RTS	
005B	A9 FF		LDA # FF	RESET SO THAT JSR DISPLAY WAITS FOR INPUT
005D	A9 FF		LDA # FF	
005D	85 0E		STA 0E	
005F	60		RTS	

COUNTER SUBROUTINE

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0060	F6 1A	INCR	INC CNTL	
0062	D0 0D		BNE UPDATE	
0064	E6 1B		INC CNTH	
0066	38		SEC	
0067	B0 08		BCS UPDATE	
0069	A5 1A	DECR	LDA CNTL	
0063	D0 02		BNE NOT	
006D	C6 1B		DEC CNTH	
006F	C6 1A	NOT	DEC CNTL	
0071	A2 1A	UPDATE	LDX #1E	
0073	20 64	FE	JSR QHEXTD1	
0076	60		RTS	

ANOTHER TRAP FOR THOSE USED TO DIFFERENT μ Ps IS THE DECIMAL FLAG. INSTEAD OF A SINGLE "DECIMAL ADJUST" INSTRUCTION TO ADJUST THE RESULT OF BINARY ARITHMETIC ON B.C.D. NUMBERS TO B.C.D. THERE ARE TWO INSTRUCTIONS

BYTES: 1 SED F8 "SET DECIMAL MODE"
1 CLD D8 "CLEAR DECIMAL MODE"

WHICH INSTRUCT THE PROCESSOR TO DO AUTOMATICALLY (OR NOT DO) THE ADJUSTMENT AFTER ARITHMETIC OPERATIONS. THIS RESULTS IN SHORTER, FASTER PROGRAMS FOR HANDLING B.C.D. ARITHMETIC WHICH, MERELY BY CHANGING THE DECIMAL MODE FLAG, WILL HANDLE BINARY ARITHMETIC. IN ORDER TO FULLY UTILISE THE μ P's POWER THE MONITOR SUBROUTINES FOR FETCHING KEYS & OUTPUTTING DATA TO THE DISPLAY HAVE BEEN WRITTEN WITHOUT ARITHMETIC SO THEY MAY BE CALLED WITH THE DECIMAL FLAG SET OR CLEARED & THEY WILL NOT AFFECT IT.

SO LET'S DO A DECIMAL ADDITION;

```
002F      F8      SED
0030      18      CLC
0031      A5 21    LDA Z 21
0033      65 20    ADC Z 20
0035      20 60 FE JSR FE60
0038      4C04 FF  JMP FF04
```

OUR STANDARD PROGRAM HAS BEEN EXTENDED BACKWARDS BY ONE BYTE, THE SED INSTRUCTION. THIS SHOULD BE INCLUDED (BY 0,0,2,F,0) G THE FIRST TIME THE PROGRAM IS RUN, BUT MAY BE OMITTED (K,0,0,3,0,4) ON SUBSEQUENT RUNS. THIS LITTLE PROGRAM WILL TELL US THAT 22 + 11 = 33, IT WILL SAY THAT 35 + 26 = 61 AND THAT 50 + 51 = 01 WHOOPS! THE PROGRAM AT FE60 ONLY DEALS WITH PUTTING THE BYTE IN THE ACCUMULATOR ON THE DISPLAY. IT PAYS NO ATTENTION TO THE CARRY FLAG, INDEED IT CHANGES THE STATE OF THE CARRY FLAG ITSELF, SO THAT WE CAN'T IMMEDIATELY CALL FE60, HAVE IT WRITE ON THE DISPLAY & RETURN THEN WRITE OUT THE STATE OF THE CARRY SOMEHOW, WHAT WE NEED IS:

I SAVE THE CARRY FLAG

II USE FE60

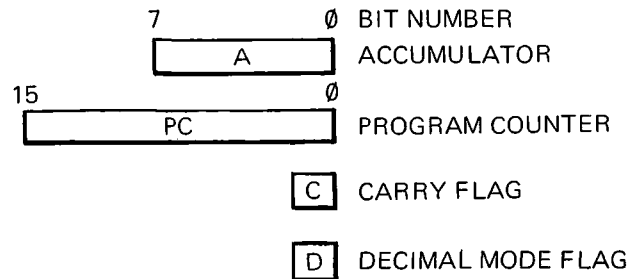
III GET THE CARRY FLAG BACK & WRITE IT OUT SOMEHOW

A FRENZIED SEARCH THROUGH THE MNEMONICS REVEALS THAT THERE ARE NO MNEMONICS LIKE LDC (LOAD C) OR STC (STORE C) A CLOSER LOOK AT THE MICROPROCESSOR IS REQUIRED.

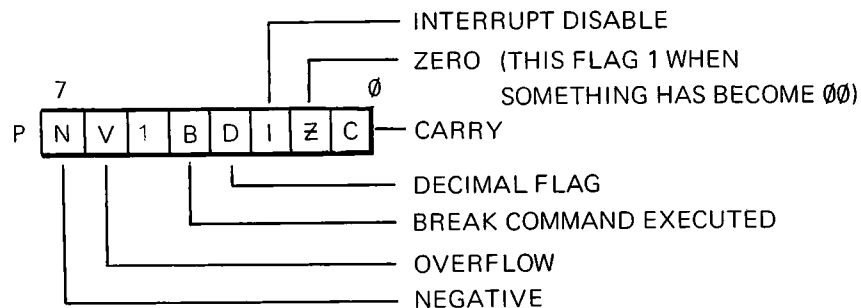
CHAPTER 3: INSIDE THE 6502

SO FAR THE PROCESSOR'S INTERNAL WORKINGS ARE

3.1 THE ACCUMULATOR, PROGRAM COUNTER, STATUS REGISTER



THE CARRY & DECIMAL MODE FLAGS HAVE BEEN TREATED SEPARATELY TO DATE. THEY ARE ACTUALLY MEMBERS OF A SPECIAL REGISTER CALLED THE PROCESSOR STATUS REGISTER, P.



CAN WE, THEN, USE LDP & STP? NO, THEY DON'T EXIST EITHER. (FUME). IN ORDER TO SOLVE THIS PROBLEM WE MUST INTRODUCE THE STACK. DID YOU WONDER JUST WHAT HAPPENED TO PC DURING A JSR? YOU WERE TOLD THAT IT WAS 'SAVED'. WHERE? HOW? IT WOULD BE TERRIBLE TO HAVE TO SPECIFY WHERE IT HAD TO BE STORED. WHAT'S NEEDED IS SOME PLACE WHERE IT CAN BE PUT DOWN AND PICKED UP AGAIN. IT WOULD BE GOOD TO ALLOW NESTED SUBROUTINES:

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0213	95 10		STA Z X 10	
0215	A9 61	INSERT	LDA #DUCK	- PUT NEW DUCK ON
0217	CA		DEX	- IN NEW POSITION
0218	10 02		BPL OLDX	- BUT NOT OVER THE END OF THE DISPLAY
021A	A2 07		LDX #07	
021C	95 10	OLDX	STA Z X 10	
021E	86 20		STX Z 20	
0220	A2 0E		LDX #TIME	- DISPLAY INTERVAL IS SET BY THE BYTE LOADED INTO X
0222	20 0C	FE WAIT	JSR DISPLAY	
0225	C5 20		CMP Z 20	- HIT?
0227	F0 05		BEQ HIT	
0229	CA		DEX	
022A	D0 F6		BNE WAIT	
022C	F0 E1		BEQ REMOVE	- FINISHED WAIT TIME
022E	A9 1C	HIT	LDA #DEAD DUCK	- PUT IN A DEAD DUCK
0230	A6 20		LDX Z 20	
0232	95 10		STA Z X 10	
0234	A9 FF		LDA #FF	
0236	85 0E		STX Z 0E	
0238	20 0C		JSR DISPLAY	- TEST FOR CONTINUATION
023B	90 C3		BCC BEGIN	
023D	4C 04	FF	JMP RESTART	- OR BACK TO THE MONITOR
023F				

MISCELLANEOUS

1

THE COUNTER PROGRAM COULD BE USED AS A SUBROUTINE IN A LONGER PROGRAM WHEN "JSR INCR" AND "JSR DECR" WOULD INCREMENT OR DECREMENT THE DISPLAY. IF THE PROGRAM APPENDED IS ALSO ENTERED THE DISPLAY WILL INCREASE OR DECREASE RAPIDLY IF "UP" OF "DOWN" KEYS ARE DEPRESSED. THIS WILL BE STOPPED BY ANY HEX KEY. IT WILL INCREMENT BY THE INDICATED AMOUNT IF KEYS 1-F ARE DEPRESSED AND WILL IGNORE ALL OTHER KEYS.

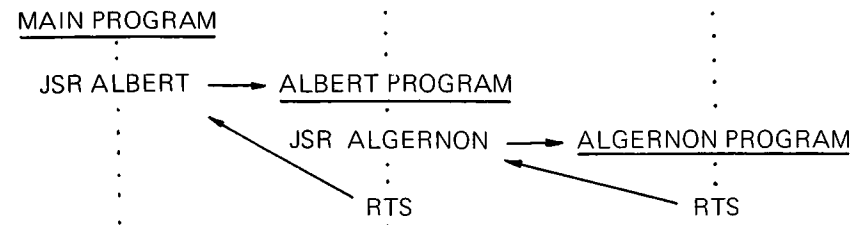
YOU SHOULD PARTICULARLY NOTICE THAT A JSR DISPLAY RETURNS WITH THE CARRY BIT CLEAR AND THE ACCUMULATOR HOLDING THE VALUE OF THE KEY PRESSED FOR THE NUMERICAL KEYS, AND THE CARRY BIT SET AND THE VALUES 0-7 IN THE ACCUMULATOR FOR THE CONTROL KEYS. IF MEMORY LOCATION 0E, WHICH IS DEDICATED TO THE MONITOR AND SHOULD NOT NORMALLY BE USED IN PROGRAMS, HAS THE MOST SIGNIFICANT BIT CLEAR THEN JSR DISPLAY WILL SCAN ONLY ONCE, IF IT IS SET IT WILL WAIT FOR A KEY TO BE DEPRESSED BEFORE RETURNING TO THE PROGRAM. IT IS A GOOD IDEA TO LOAD IT WITH 'IF' IF YOU WISH TO USE THIS FACILITY AS OTHER VALUES MAY CAUSE YOU DIFFERENT PROBLEMS. AGAIN SEE THE REST OF THIS MANUAL IF YOU REALLY WISH TO UNDERSTAND THE PROCESS.

ADDR	HEX	LABEL	INSTRUCTION	COMMENTS
	CODE			
027C	69 00		ADC #00	
027E	46 2A		LSR ANAL + 2	
0280	69 00		ADC #00	
0282	46 2B		LSR ANAL + 3	
0284	69 00		ADC #00	
0286	4A		LSRA	
0287	B0 BF		BCS ONEOFF	- NOT A GOOD MOVE
0289	C6 1F		DEC COUNT	
028B	D0 E8		BNE CONT	- KEEP CHECKING THE MOVE
028D	A2 03		LDX #03	- GOOD MOVE, TRANSFER TO ACTUAL STACKS
028F	B5 24	BAT	LDAZX POSS	
0291	95 20		STAZX STACK	
0293	CA		DEX	
0294	10 F9		BPL BAT	
0296	4C 00	02	JMP HUMMOV	- OPPONENT.
0299	A9 00	DSPGAP	LDA #00	
029B	A2 07		LDX #07	
029D	95 10	CLEAR	STAZX D	- CLEAR THE DISPLAY FIRST
029F	CA		DEX	
02A0	10 FB		BPL CLEAR	
02A2	D8		CLD	- CLEAR DECIMAL MODE
02A3	A2 04		LDX #04	- DISPLAY STACKS
02A5	A0 01 07		LDY #01 07	
02AY	B5 1F	AROUND	LDAZX STACK -1	
02A9	20 7A FE		JSR HEXTD	
02AC	08 38		INY DE Y	
02AD	08 38		INY DE Y	
02AE	CA		DEX	
02AF	D0 F6		BNE AROUND	
02B1	60		RTS	
02B2				

2
THE DUCKSHOOT GAME IS A SPEED TEST: YOU HAVE TO SHOOT THE FLYING DUCKS. THEY SUCCESSIVELY ENTER FROM THE RIGHT AND FLY TOWARDS THE LEFT AT A SET SPEED. YOU SHOOT A DUCK BY PRESSING ITS CURRENT POSITION ON THE KEYBOARD. THE LEFT MOST DISPLAY IS 0, THE RIGHTMOST DISPLAY IS 7. WHEN A DUCK IS HIT IT DIES. THE GAME MAY BE RESTARTED WITH ANY HEX DIGIT KEY

DUCK SHOOT

ADDR	HEX	LABEL	INSTRUCTION	COMMENTS
	CODE			
0200	A9 1F	BEGIN	LDA #1F	- SINGLE SCAN DISPLAY ROUTINE
0202	85 0E		STA Z 0E	
0204	A9 00		LDA #00	- CLEAR THE DISPLAY
0206	A2 07		LDX #07	
0208	86 20		STX Z 20	
020A	95 10	CLEAR	STA Z X 10	
020C	CA		DEX	
020D	10 FB		BPL CLEAR	
020F	A9 00	REMOVE	LDA #00	- TAKE THE OLD DUCK OFF
0211	A6 20		LDX Z 20	



WE CAN'T JUST SAY THAT PC IS TO BE SAVED IN LOCATION, SAY, L & M - WE WOULDN'T GET BACK FROM ALBERT SINCE THE CALL TO ALGERNON WOULD HAVE DESTROYED THE NECESSARY INFORMATION IN L & M. (IT IS WORTH NOTING HERE THAT L & M COULD BE "CALLED" -2 "CALLED" -1. THEN A CALL TO ALBERT AS A SUBROUTINE WOULD STORE THE RETURN ADDRESS JUST BEFORE THE START OF ALBERT ALLOWING NESTED SUBROUTINES AS ABOVE. A PROBLEM IS THAT THIS DOES NOT WORK WITH READ ONLY MEMORY, LIKE THE MONITOR).

3.2 THE STACK POINTER

WE NEED SOMETHING WHICH WILL DECIDE WHAT L & M ARE TO BE, DEPENDING ON WHICH SUBROUTINE WE ARE IN. AN OBVIOUS CHOICE IS TO USE AN ARRAY OF MEMORY LOCATIONS, AND A VARIABLE WHICH POINTS TO THE CURRENT LOCATION OF L & M. EACH TIME WE DO A JSR WE STEP UP THE POINTER & EACH TIME WE DO AN RTS WE STEP IT DOWN.

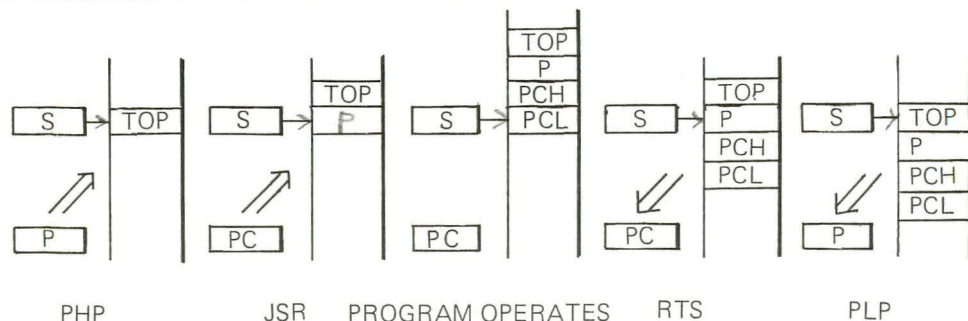


WITH ACORN WE'LL NEED TWO BYTES FOR EACH RETURN ADDRESS. THIS IS NO TROUBLE, WE JUST INCREMENT & DECREMENT THE POINTER TWICE. THE WHOLE PROCESS IS CARRIED OUT BY THE PROCESSOR AUTOMATICALLY ON EACH JSR & RTS, THE POINTER IS CALLED THE STACK POINTER AND IS A SPECIAL 8 BIT REGISTER INSIDE THE PROCESSOR. THE ARRAY IS USUALLY CALLED A STACK SINCE IT CAN ALSO BE USED TO STORE THINGS OTHER THAN RETURN ADDRESSES. THE ACTUAL STACK RUNS FROM 01FF DOWN TO 0100, AND IT STARTS AT THE TOP: AN EMPTY STACK HAS STACK POINTER AT FF. A BYTE IS PUT ON THE STACK AND THE POINTER IS DECREMENTED TO POINT AT THE NEXT LOCATION; THE POINTER IS INCREMENTED AND A BYTE LOADED FROM THE STACK IN THE REVERSE OPERATION. NO CHECK IS MADE FOR THE 00 TO FF DECREMENT INDICATING AN OVERFLOWED STACK, SO PROGRAMS THAT REQUIRE MORE THAN 256 BYTES OF STACK SPACE WILL MYSTERIOUSLY FAIL. SINCE THIS IS 128 CONSECUTIVE JSR'S, THE PROBLEM WON'T BE ENCOUNTERED VERY OFTEN. . .
NOW THE PROCESSOR STATUS REGISTER CAN BE PUSHED ONTO THE STACK:

PLP 28 "PULL P"
 PHP 08 "PUSH P"
 AND SO WE MAY SAVE IT BEFORE A SUBROUTINE CALL AND RECOVER IT
 AFTERWARDS

PHP
 JSR . . .
 PLP

THE SEQUENCE OF STACK OPERATIONS IS



SO WE HAVE NOW MANAGED TO SAVE THE CARRY FLAG, USE FE60, AND
 REGAIN THE CARRY FLAG. WE WISH TO WRITE IT OUT, SO IT WOULD HAVE
 BEEN BETTER TO WRITE.

PHP
 JSR FE60

PLA PULL BYTE FROM STACK INTO A

SINCE THIS GIVES THE CARRY FLAG IN A, AS THE LEAST SIGNIFICANT BIT,
 TO GET RID OF THE REST OF THE BITS OF THE RECOVERED STATUS
 REGISTER, WE CAN SIMPLY AND # 01. NOW A CONTAINS 0 OR 1 DEPENDING
 ON THE CARRY FROM ORIGINAL SUM. OUR PROGRAM NOW IS

SED SET UP FOR DECIMAL ADD

CLC

LDA Z 21 DO IT

ADC Z 20

PHP

JSR FE60 SAVE CARRY

WRITE OUT TWO DIGITS
 ON DISPLAYS 6 & 7

PLA

AND # 01

A = 0 (NO CARRY FROM SUM)
 OR A = 1 (CARRY FROM SUM)

NOW ALL WE NEED TO DO IS WRITE OUT THE ACCUMULATOR ON DISPLAY
 NO.5. THE WAY WE WROTE OUT THE FIRST TWO DIGITS OF THE RESULT WAS
 TO USE A MONITOR SUBROUTINE WHICH DID JUST THAT. YOU'VE PROBABLY
 NOTICED THAT THE MONITOR ONLY PUTS A DOT ON DISPLAY 5 (THE 3RD

ADDR	HEX		LABEL	INSTRUCTION	COMMENTS
	CODE				
0210	29 7F			AND #7F	
0212	95 11			STAZX D + 1	
0214	E8			INX	— MOVE FORWARD
0215	E8			INX	
0216	E0 07			CPX #07	— END OF STACKS?
0218	90 E9			BCC SHIFTP	
021A	A2 00			LDX #00	
021C	F0 E5			BEQ SHIFTP	
021E	A8		MINUS	TAY	
021F	F0 E8			BEQ CHEAT	— PREVENT ZERO FROM BEING USED
0221	8A			TXA	
0222	4A			LSRA	— ADDRESS OF REQUIRED STACK
0223	AA			TAX	
0224	38			SEC	
0225	B5 20			LDAZX STACK	— DO THE PLAYER'S MOVE
0227	E5 0D			SBC KEY	
0229	95 20			STAZX STACK	
022B	20 99	02	COMMOV	JSR DSPGAP	— SHOW STACKS
022E	84 0E			STY REPEAT	
0230	A2 00			LDX #00	
0232	20 0C	FE	WAIT	JSR DISPLAY	— THINKING TIME
0235	CA			DEX	
0236	D0 FA			BNE WAIT	
0238	CA			DEX	
0239	86 0E			STX REPEAT	— CLEAR REPEAT STATUS
023B	A0 03			LDY #03	
023D	A2 03		NEXTS	LDX #03	— TRANSFER STACK TO POSS
023F	B5 20		BLOCK	LDAZX STACK	— POSS REPRESENTS THE POSSIBLE COMPUTER
					— MOVES
0241	95 24			STAZX POSS	
0243	CA			DEX	
0244	10 F9			BPL BLOCK	
0246	A2 03		ONEOFF	LDX #03	— TRANSFER POSS TO ANAL
0248	B5 24		BRICK	LDAZX POSS	— ANAL REPRESENTS THE MOVE BEING
					— ANALYSED
024A	95 28			STAZX ANAL	
024C	CA			DEX	
024D	10 F9			BPL BRICK	
024F	A2 03			LDX #03	
0251	B9 24	00		LDA, Y POSS	
0254	38			SEC	
0255	E9 00	01		SBC #01	
0257	99 24	00		STA, Y POSS	— POSS CONTAINS POSSIBLE MOVE
025A	99 28	00		STA, Y ANAL	— ANAL CONTAINS POSSIBLE MOVE
025D	B0 12			BCS CHECK	
025F	88			DEY	
0260	10 DB			BPL NEXTS	— TRY ALL STACKS
0262	B5 20		TRY	LDAZX STACK	— CHECK IF STACK EMPTY
0264	F0 05			BEQ EMPTY	
0266	D6 20			DECZX STACK	— MAKE DESPERATE MOVE
0268	4C 00	02		JMP HUMMOV	
026B	CA		EMPTY	DEX	
026C	10 F4			BPL TRY	
026E	4C 04	FF		JMP RESTART	— LOST.
0271	A9 04		CHECK	LDA #04	
0273	85 1F			STA COUNT	
0275	A9 00		CONT	LDA #00	— EVALUATE MOVE
0277	46 28			LSR ANAL	
0279	2A			ROLA	
027A	46 29			LSR ANAL + 1	

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0207	E0 07		CPX #07	
0209	D0 F7		BNE LOOP	- KEEP GOING
020B	85 17		STA Z D + 7	- NEW DATA
020D	60		RTS.	
020E				

GAMES PROGRAMS

1

NIM IS A TRADITIONAL GAME IN WHICH THE PLAYERS ALTERNATIVELY REMOVE STICKS, OR COINS, OR WHATEVER FROM ONE OF SEVERAL STACKS. THE ONLY RULES ARE THAT YOU MUST TAKE AT LEAST ONE PIECE PER MOVE AND THAT YOU CAN ONLY REMOVE PIECES FROM ONE STACK PER MOVE. THERE IS A WELL-DEFINED STRATEGY FOR OPTIMAL PLAY BUT THIS DOES NOT GUARANTEE A WIN UNLESS THE OPPONENT MAKES A MISTAKE OR THE INITIAL SITUATION IS AGAINST HIM. THE COMPUTER PLAYS WELL BUT, WITH LUCK, CAN BE BEATEN. THE WINNER IS THE PLAYER WHO REMOVES THE LAST PIECE

IN THIS VERSION OF THE GAME THERE ARE FOUR STACKS OF FROM 0-F PIECES. YOU MUST ENTER THE SIZE OF YOUR STACKS IN MEMORY LOCATIONS 20-23 BEFORE STARTING THE GAME. THE GAME STARTS AT 002F AND YOUR MOVE OR 0180 AND THE COMPUTER'S MOVE. ON RUNNING, THE DISPLAY WILL SHOW A · B C D WHERE A,B,C,D ARE THE CONTENTS OF THE STACKS. ANY CONTROL KEY WILL MOVE THE POINTER (FULL STOP) AROUND THE STACKS. WHEN IT POINTS TO THE STACK FROM WHICH YOU WISH TO REMOVE PIECES PRESS THE KEY CORRESPONDING TO THE NUMBER YOU WISH TO REMOVE. ZERO IS ILLEGAL AND WILL NOT BE ALLOWED. IF YOU SUBTRACT MORE PIECES THAN ARE IN THE STACK THE GAME WILL GET VERY CONFUSED.

AFTER REMOVAL OF PIECES THE DISPLAY WILL SHOW THE CURRENT SITUATION AND THE COMPUTER WILL MAKE ITS MOVE.

CONTINUE UNTIL SOMEONE (SOMETHING?) WINS.

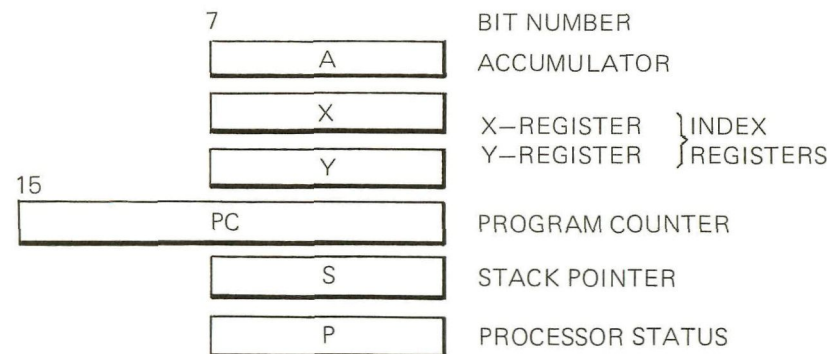
YOU MIGHT LIKE TO TRY AND WRITE SUBROUTINES TO PRINT MESSAGES ON THE DISPLAY IN THE EVENT OF EITHER A HUMAN OR COMPUTER VICTORY. A CHECK WOULD HAVE TO BE INSERTED TO DECIDE A COMPUTER WIN BUT THE JUMP FOR A HUMAN WIN IS ALREADY THERE UNDER THE MNEMONIC JMP MESSAGE, THOUGH THE CODE IN FACT JUMPS TO THE HUMAN MOVE.

NIM					NOT RELOCATABLE
0200	20 99 02	HUMMOV	JSR DSPGAP		- CLEAR DECIMAL
0203	B5 11	SHIFTPT	LDZX D + 1		- DISPLAY STACKS
0205	09 80		ORA #80		- SET DECIMAL POINT ON
0207	95 11		STAZX D + 1		
0209	20 0C	FE CHEAT	JSR DISPLAY		- WAIT FOR INPUT
020C	90 10		BCC MINUS		
020E	B5 11		LDAZX D + 1		- REMOVE CURRENT DECIMAL POINT

FROM THE RIGHT) AND SUSPECT THAT IT CAN'T PUT ANYTHING ELSE THERE. THIS IS TRUE, BUT IT DOESN'T MEAN THAT THERE ISN'T A MONITOR SUBROUTINE THAT CAN DO THE JOB. SUCH A SUBROUTINE LIVES AT FE7A. IT IS DESIGNED TO PUT THE LOWEST FOUR BITS OF THE ACCUMULATOR ONTO ANY OF THE DISPLAYS, AS A READABLE CHARACTER. THIS IS JUST WHAT WE NEED - BUT HOW DO WE TELL THE SUBROUTINE WHICH DISPLAY TO USE?

3.3 THE INTERNAL REGISTERS X AND Y.

WELL, BACK TO THE μ P. THIS IS WHAT IT LOOKS LIKE INSIDE



TWO NEWCOMERS, YOU'LL NOTICE! X & Y ARE 'INDEX REGISTERS', THEY WILL BE DEALT WITH MORE THOROUGHLY IN A FEW MORE PAGES, BUT WHAT MATTERS NOW IS THE USE FE7A MAKES OF THEM:

I FE7A NEITHER CARES ABOUT, NOR CHANGES X

II FE7A DOESN'T CHANGE Y, BUT THE DISPLAY IT PUTS A ONTO IS CONTROLLED BY Y THAT IS, THE LOWER 4 BITS OF A ARE TRANSFORMED INTO THE CORRECT SEQUENCE OF BITS TO REPRESENT THEIR HEXADECIMAL CHARACTER AS IT SHOULD APPEAR ON THE 7 SEGMENT DISPLAY. THEN THIS IS STORED IN MEMORY TO AWAIT THE SUBROUTINE WHICH ACTUALLY PUTS THINGS ON DISPLAY.

ALTHOUGH FE7A MAKES NO RESTRICTIONS ON THE SIZE OF Y, THE MONITOR SUBROUTINE WHICH DISPLAYS THEM ONLY KNOWS ABOUT THE FIRST 8 (NUMBERED, OF COURSE, 0-7) OF THEM, IN LINE WITH THE ACTUAL DISPLAY HARDWARE. DISPLAY 0 IS THE LEFTMOST, DISPLAY 7 IS THE RIGHTMOST. TO KEEP THE MONITOR AS EFFICIENT AS POSSIBLE THE SUBROUTINE AT FE60 USES FE7A. IT FOLLOWS THAT IT MUST HAVE LOADED Y WITH 7 & 6, AND SINCE FE7A DOESN'T CHANGE Y, Y IS STILL SET TO THE LAST USED OF THESE WHICH IS 6. SO. INSTEAD OF USING

LDY #05 A0 05 "LOAD Y WITH THE NEXT BYTE" (05 HERE)
WE CAN USE
DEY 88 "DECREMENT (IN HEXADECIMAL) Y BY ONE"

TO SET Y TO 5, THUS SAVING A WHOLE BYTE! (BUT NO TIME, THE TWO INSTRUCTIONS ARE EXECUTED IN THE SAME TIME, $2\mu s$). THE COMPLETE PROGRAM IS

```
002F F8      SED
0030 18      CLC
0031 A5 21   LDA Z 21
0033 65 20   ADC Z 20
0035 08      PHP
0036 20 60 FE JSR FE60
0039 68      PLA
003A 29 01   AND # 01
003C 88      DEY
003D 20 7A FE JSR FE7A
0040 4C 04 FF JMP FF04
```

AND SO, AT LAST, WE FIND THE ANSWER TO $50_{10} + 50_{10}$ IS

K. 002F 100

PERHAPS WE SHOULD HAVE CLEARED THE DISPLAY? OR MADE IT SHOW THE NUMBERS TO BE ADDED TOGETHER? OR ACTUALLY FETCHED THE TWO NUMBERS USING KEYBOARD AND DISPLAY LIKE THE MONITOR DOES? OR SOME COMBINATION OF THESE?

3.4 MAKING OUR PROGRAM 'FRIENDLY'

USING THE MONITOR SUBROUTINE AT FE88 IT IS EASY TO DO THE THIRD OPTION. FE88 IS THE ROUTINE WHICH FETCHES 4 DIGIT NUMBERS, TERMINATED BY ANY COMMAND KEY, INTO THE TWO BYTES IN ZERO PAGE X & X + 1 [i.e. IF X CONTAINS 20, INTO 0020 (LOW BYTE = RH PAIR OF NUMBERS) & 0021] JUST WHAT WE NEED!

```
002A F8      SED
002B A2 20   LDX # 20
002D 20 88 FE JSR FE 88
0030 18      CLC
0031 A5 21   LDA Z 21
0033 65 20   ADC Z 20
0035 08      PHP
0036 20 60 FE JSR FE60
0039 68      PLA
003A 29 01   AND # 01
003C 88      DEY
003D 20 7A FE JSR FE7A
0040 4C 04 FF JMP FF04
```

ONCE AGAIN THE PROGRAM HAS BEEN EXTENDED BACKWARDS SINCE THE GREATER PART OF IT HAS ALREADY BEEN ENTERED (UNLESS YOU'VE SWITCHED OFF AND LOST IT ALL)

RUNNING THIS PROGRAM (G0,0,2,A, k) PRODUCES

K. 5050 . (ON THE ASSUMPTION THAT 0020 & 0021 STILL CONTAIN THE 50'S ADDED TOGETHER AS BEFORE)

THE FIRST PROGRAM, TEST, IS TRIVIAL: IT JUST SENDS A PARTICULAR BYTE TO TAPE REPETITIVELY. IT MUST BE STOPPED BY RESET. RECORD A FEW MINUTES OF THIS, THEN LOAD IT USING LOAD. DEVIATIONS FROM THE STATIONARY PATTERN ARE EASY TO SEE. THE SECOND PROGRAM, RETAG, IS RELOCATABLE. IT ACTS JUST LIKE THE MONITOR'S STORE ROUTINE, EXCEPT THAT IT ASKS FOR AN EXTRA ADDRESS. THE DATA WHICH IS STORED IS THAT STARTING AT THIS LAST ADDRESS, IT PRETENDS TO BE SITUATED BETWEEN THE FIRST TWO ADDRESSES. INCORPORATE THE REQUIRED STATE OF ZERO PAGE REGISTERS IN FRONT OF YOUR DATA, THEN 'LOAD AND AUTO RUN' PROGRAMS MAY BE CREATED.

TAPE PROGRAMS

NOT RELOCATABLE

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0200	A9 55	TEST	LDA #55	- THE TEST BYTE
0202	20 B1 FE		JSR PUTBYTE	- SEND IT
0205	4C 00 02		JMP TEST	- KEEP SENDING IT
0208	A9 F1	RETAG	LDA #F1	- F. PROMPT
020A	85 10		STA D	
020C	A2 06		LDX #06	
020E	20 88 FE		JSR QDATFET	- FIRST ADDRESS
0211	A2 08		LDX #08	
0213	86 10		STX D	- PROMPT
0215	20 88 FE		JSR QDATFET	- SECOND ADDRESS
0218	A9 46		LDA #46	- PROMPT
021A	85 10		STA D	
021C	A2 20		LDX #20	
021E	20 88 FE		JSR QDATFE7	- LAST ADDRESS: ACTUAL DATA START
0221	A2 04		LDX #04	
0223	B5 05	ADRSS	LDA Z,X 05	- SEND FAKE ADDRESSES
0225	20 B1 FE		JSR PUTBYTE	
0228	CA		DEX	
0229	D0 F8		BNE ADDRSS	
022B	A0 00	DATAS	LDY #00	
022D	B1 20		LDA (20),Y	- PROPER DATA
022F	E6 20		INC 20	- INCREMENT PROPER DATA COUNTER
0231	D0 02		BNE NOINC	
0233	E6 21		INC 21	
0235	20 B1 FE	NOI1NC	JSR PUTBYTE	- SEND DATA
0238	20 A0 FE		JSR COM16	- CHECK FAKE ADDRESSES FOR END
023B	D0 EE		BNE DATAS	
023D	4C 04 FF		JMP RESTART	
0240				

THE SCROLL PROGRAM SHIFTS THE WHOLE DISPLAY ONE LEFT, AND ENTERS THE NEW INFORMATION, IN A, ON THE FAR RIGHT.

SCROLL

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0200	A2 00		LDX #00	- MUST GO FORWARDS
0202	B4 11	LOOP	LDY ZX D + 1	- PICK-UP DATA ON RIGHT
0204	94 10		STY ZX D	- & MOVE IT ONE LEFT
0206	E8		INX	

025A 00 71 77
 025D 50 00
 025F

THE RELOCATOR FIRST FETCHES THE THREE ADDRESSES IT REQUIRES, THE ADDRESSES OF THE START & END OF THE MEMORY SECTION TO BE MOVED, AND THE ADDRESS OF THE START OF THE AREA TO WHICH THE MOVE IS TO TAKE PLACE. THE PROMPTS ARE F., & t RESPECTIVELY. AFTER TERMINATING THE LAST ADDRESS, THE MOVE TAKES PLACE. MOVES UP BY LESS THAN THE LENGTH OF THE MATERIAL TO BE USED WILL NOT BE SUCCESSFUL (I.E. t - F ., IF POSITIVE, SHOULD BE GREATER THAN -t)

RELOCATOR

ADDR	HEX		LABEL	INSTRUCTION	COMMENTS
	CODE				
0200	A2 F1			LDX #F1	
0202	86 10			STX Z D	- SET UP FROM PROMPT F.
0204	A2 20			LDX #20	
0206	20 88	FE		JSR QDATFET	- AND GET ADDRESS
0209	A2 46			LDX #46	
020B	86 10			STX Z D	- SET UP END PROMPT
020D	A2 22			LDX #22	
020F	20 88	FE		JSR QDATFET	- AND GET SECOND ADDRESS - MOVE THE DATA BETWEEN THESE ADDRESSES
0212	A2 78			LDX #78	
0214	86 10			STX Z D	- SET UP TO PROMPT
0216	A2 24			LDX #24	
0218	20 88	FE		JSR QDATFET	- AND GET BASE ADDRESS - MOVE TO HERE & SUCCESSIVE LOCATIONS
021B	A2 1A			LDX #1A	
021D	A1 06		MOVE	LDA (06,X)	- DO THE MOVE
021F	91 24			STA (24,Y)	
0221	C8			INY	- INCREMENT THE TO ADDRESS
0222	D0 02			BNE NOINC	
0224	E6 25			INC Z 25	
0226	20 A0	FE	NOINC	JSR COM16	- USE COM16 TO DO THE LIMIT TEST
0229	D0 F2			BNE MOVE	
022B	4C 04	FF		JMP RESTART	
022D					

YOU SHOULD ENTER THE TWO PAIRS OF NUMBERS YOU WISH ADDED TOGETHER AS IF THEY FORMED AN ADDRESS. TERMINATING YOUR ENTRY WITH k INSTANTLY PRODUCES THE RESULT

K. 5050 100

LOOKING BACK OVER THE PROGRAM, AND EXAMINING THE MONITOR LISTING, WILL REVEAL THAT IT TOOK AD₁₆ (OR 173₁₀) BYTES OF CODE TO ACHIEVE THIS. THE ACTUAL OPERATION USED 6 BYTES OF CODE (SED; CLC; LDA Z; ADC Z) WHILE THE OTHER 167₁₀ ARE THERE 'MERELY' TO DISPLAY THE RESULT & FETCH THE INFORMATION NEATLY (THE CODE CALCULATIONS DO NOT CONSIDER THE 16₁₀ BYTES OF CHARACTER FONT OR THE 11₁₀ BYTES OF TEMPORARY STORAGE ALSO USED)

CHAPTER 4: THE REMAINDER OF THE INSTRUCTION SET

4.1 BRANCHES

THINKING ABOUT THE FE88 PROGRAM, YOU SHOULD REALIZE THAT IT DOES SOMETHING OF THE FORM

FETCH NEXT KEY

IF KEY IS A COMMAND KEY THEN RETURN

THIS IS A CONDITIONAL TRANSFER OF CONTROL AND REPRESENTS SOME NEW INSTRUCTIONS AND A DIFFERENT WAY OF CHANGING THE PROGRAM COUNTER. AN OPERATION LIKE ADC DOES MORE THAN ADDING TWO BYTES AND THE CARRY FLAG TOGETHER AND OUTPUTTING A CARRY. IT ALSO SETS SOME OF THE OTHER FLAGS IN P:

THE Z FLAG IS SET IF THE RESULTING BYTE WAS ZERO

THE V FLAG IS SET IF THERE WAS A 2'S COMPLEMENT OVERFLOW

THE N FLAG IS SET IF THE RESULT WAS A NEGATIVE 2'S COMPLEMENT NUMBER - I.E. BECOMES BIT 7 OF THE RESULT.

THESE FLAGS ARE ABLE TO CAUSE CONDITIONAL TRANSFER BY USING THE APPROPRIATE ONE OF THE EIGHT 'BRANCH' INSTRUCTIONS. THE MECHANISM EMPLOYED IS TO PERFORM A 2'S COMPLEMENT ADD BETWEEN THE PROGRAM COUNTER AND THE SECOND BYTE OF THE BRANCH INSTRUCTION THUS PERMITTING THE TRANSFER TO BE -128. . . +127 BYTES FROM THE NEXT INSTRUCTION. THIS IS CALLED 'RELATIVE ADDRESSING' AND IS A POSITION INDEPENDENT METHOD OF TRANSFER, THE EIGHT BRANCH INSTRUCTIONS ARE ASSOCIATED TWO TO EACH OF THE C, Z, V & N FLAGS, ONE OF WHICH BRANCHES IF THE FLAG IS SET, THE OTHER BRANCHES IF IT IS CLEAR.

TO CLARIFY THIS LET'S LOOK AT AN EXAMPLE:

```

*+0      BCS 03      "BRANCH IF CARRY SET"
*+2      SEC        SET CARRY
*+3      BCS 01
*+5      CLC        CLEAR CARRY
*+6      . . . . .
  
```

(THE ARROWS ARE PUT IN FOR CLARITY)

WE'LL NEED TO CONSIDER THIS PROGRAM BOTH WITH THE CARRY SET & WITH IT CLEAR

| CARRY IS CLEAR

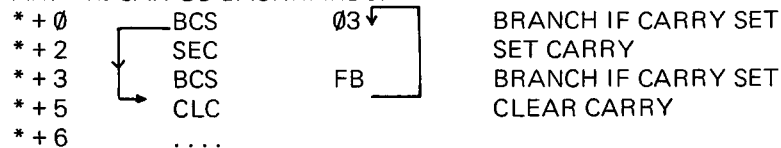
INSTRUCTION I DOES NOT TRANSFER CONTROL SO WE DO INSTRUCTION II, SEC, NOW INSTRUCTION III TRANSFERS CONTROL SINCE THE CARRY IS NOW SET. 01 IS ADDED TO THE PC (= * + 5) TO GIVE * + 6 AS THE ADDRESS OF THE NEXT INSTRUCTION.

II CARRY IS SET

INSTRUCTION I TRANSFERS CONTROL. 03 IS ADDED TO THE PC (= * + 2) TO GIVE * + 5 AS THE ADDRESS OF THE NEXT INSTRUCTION, INSTRUCTION IV. CLC.

SO IF THE CARRY WAS CLEAR IT IS SET; IF IT WAS SET IT IS CLEARED, SO THE PROGRAM COMPLEMENTS THE CARRY (THERE ARE QUICKER METHODS, INDEED IT CAN BE DONE WITH 3 INSTRUCTIONS IN 4 BYTES).

AND WE CAN GO BACKWARDS:



IF THE CARRY IS SET THE PROGRAM IS AS BEFORE IF IT IS CLEARED WE SET IT & BRANCH FB

$$\left(\begin{array}{r} 2\text{'s COMPLEMENT ADD} \\ * + 5 \\ \text{FB} \\ * + 0 \end{array} \right) +$$

—BACK TO THE BEGINNING. A RATHER COMPLICATED WAY OF CLEARING THE CARRY.

MOST OF THE NON-BRANCH INSTRUCTIONS WILL CHANGE SOME OF THESE 4 TESTABLE FLAGS, USUALLY THE N & Z FLAGS SINCE THEY CONSTANTLY MONITOR THE STATUS OF OPERANDS SO BRANCHES WILL APPEAR RATHER FREQUENTLY IN PROGRAMS.

4.2 INDEXING

IF YOU WISHED TO CLEAR (SET EACH BYTE TO 0) A PATCH OF MEMORY, e.g. THE MEMORY USED TO STORE THE DATA WHICH IS TO BE OUTPUT TO THE DISPLAYS, WHICH IS FROM 0010 TO 0017, YOU MIGHT THINK

LDA# 00	LOAD ACCUMULATOR IMMEDIATE WITH 00
STA Z 10	STORE ACCUMULATOR IN ADDRESS 0010
STA Z 11	STORE ACCUMULATOR IN ADDRESS 0011
STA Z 12	STORE ACCUMULATOR IN ADDRESS 0012
:	
STA Z 17	STORE ACCUMULATOR IN ADDRESS 0017

IS NECESSARY. THIS LOOKS SUFFICIENTLY REGULAR THAT THE COMPUTER SHOULD BE ABLE TO DO IT. THIS IS WHERE THE INDEX REGISTERS REAPPEAR. WE CAN STORE THE ACCUMULATOR INDEXED BY EITHER INDEX REGISTER

STA Z,X	95	"STORE A INDEXED BY X IN ZERO PAGE"
STA Z,X 10		

THE OFFSET CALCULATOR CALCULATES THE OFFSET TO BE ENTERED AS THE SECOND BYTE OF A BRANCH INSTRUCTION. IT WILL PROMPT WITH XX0000XX AND YOU SHOULD ENTER THE ADDRESS OF THE BRANCH INSTRUCTION. AFTER A CONTROL KEY IT WILL PROMPT AGAIN WITH XX1111XX AND YOU SHOULD ENTER THE ADDRESS YOU WISH TO BRANCH TO. THE REPLY WILL BE EITHER "OFFSET XX" WHERE XX IS THE VALUE TO BE ENTERED, OR "TOO FAR" IF THAT IS THE CASE. A CONTROL KEY RESTARTS THE SEQUENCE.

OFFSET CALCULATOR

NOT RELOCATABLE

ADDR	HEX		LABEL	INSTRUCTION	COMMENTS
	CODE				
0200	D8			CLD	
0201	A9 02	AGAIN		LDA #02	
0203	85 21			STA MESSH	— INITIALIZE MESSAGE POINTER
0205	84 22			STY FROMH	— SET UP PROMPT
0207	84 23			STY FROML	
0209	A2 22			LDX #FROML	
020B	20 88	FE		JSR QDATFET	— FETCH FIRST ADDRESS
020E	A9 11			LDA #11	— SET UP 2ND PROMPT
0210	85 24			STA TOL	
0212	85 25			STA TOH	
0214	A2 24			LDX #TOL	
0216	20 88	FE		JSR QDATFET	— FETCH SECOND ADDRESS
0219	A5 22			LDA FROML	— OFFSET TO MAKE OVERLENGTH EASY
021B	E9 7E			SBC #7E	— CARRY KNOWN SET BY QDATFET
021D	85 22			STA FROML	
021F	B0 03			BCS HSUB	— DON'T SET THE CARRY AGAIN!
0221	C6 23			DEC FROMH	
0223	38			SEC	
0224	A5 24	HSUB		LDA TOL	— CALCULATE THE LENGTH
0226	E5 22			SBC FROML	
0228	AA			TAX	
0229	A5 25			LDA TOH	
022B	E5 23			SBC FROMH	
022D	D0 0E			BNE TOOFAR	
022F	A9 51			LDA #51	
0231	20 44	02		JSR MESSAGE	— PRINT OUT
0232	8A			TXA	
0235	49 80			EOR #80	— COMPLEMENT TOP BIT BECAUSE OF THE OFFSET APPLIED
0236	20 60	FE		JSR RDHEXTD	— PRINT OUT ANSWER, OVER WRITING THE
0239	4C 04	FF		JMP RESTART	— FINISHED
023C	A9 57	TOOFAR		LDA #57	— WHOOPS
023E	20 44	02		JSR MESSAGE	— TELL THE PROGRAMMER THAT IT'S WRONG
0241	4C 01	02		JMP AGAIN	— AND GET IT DONE AGAIN
0244	85 20	MESSAGE		STA MESSL	— MESSAGE DESCRIBED BY A
0246	A0 07			LDY #07	— EIGHT BYTES OF DATA TO DISPLAY
0248	B1 20	LOOP		LDA (MESSL), Y	— FETCH THEM
024A	99 10	00		STA D, Y	
024D	88			DEY	
024E	10 F8			BPL LOOP	
0250	60			RTS	
0251	5C 71	71			— THE DATA
0254	ED 79	78			
0257	78 5C	5C			

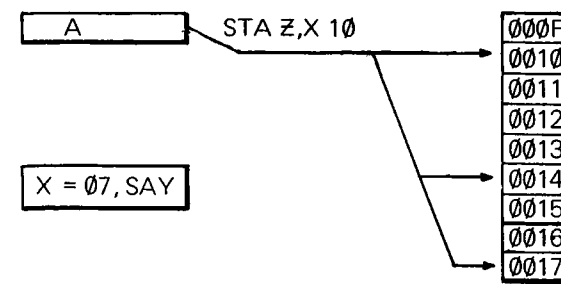
THE HEXADEXIMAL TO DECIMAL CONVERTER PROMPTS WITH XX0000XX AND AFTER A CONTROL KEY IS PRESSED WILL PROVIDE AN ANSWER IN THE FORM ????????, AFTER A WAIT!

THE PROGRAM WORKS BY DECREMENTING THE HEX. NUMBER AND INCREMENTING THE DECIMAL NUMBER UNTIL THE HEX. NUMBER REACHES ZERO.

THIS PROGRAM, LIKE THE DECIMAL TO HEX. CONVERTER, WHICH USES VIRTUALLY THE SAME METHOD, ILLUSTRATES THE USE OF THE DECIMAL MODE, AN IMPORTANT FACET OF THIS PROCESSOR. THEY ALSO PROVIDE AN EXCELLENT DEMONSTRATION OF THE TRADEOFF FREQUENTLY FOUND BETWEEN PROGRAM LENGTH AND SIMPLICITY, AND PROGRAM EXECUTION TIME. THE METHOD USED IS BOTH SHORT AND SIMPLE, BUT CAN TAKE UP TO THREE SECONDS FOR SOME CALCULATIONS. A MUCH LONGER AND MORE COMPLEX (RELATIVELY) PROGRAM COULD HAVE BEEN WRITTEN BASED ON $ABCD = A(16 \times 16 \times 16) + B(16 \times 16) + C(16) + D$ AND WOULD HAVE BEEN VIRTUALLY INSTANTANEOUS.

HEX → DEC

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0200	84 20		STY Z HEXL	— SET UP ZERO PROMPT
0202	84 21		STY Z HEXH	—
0204	A2 20		LDX #HEXL	
0206	20 88	FE	JSR Q DATFET	— AND FETCH THE DATA
0209	F8		SED	— DECIMAL MODE
020A	A2 00		LDX #00	— SET X & Y & DECOUT TO ZERO
020C	86 22		STX Z DECOUT	
020E	A5 20	DECRHEX	LDA Z HEXL	— TEST FOR ZERO, THEN DECREMENT
0210	D0 06		BNE NODEL	
0212	A5 21		LDA Z HEXH	
0214	F0 13		BEQ DEAD	— IF HEX NO. IS ZERO, THEN FINISHED
0216	C6 21		DEC Z HEXH	
0218	C6 20	NODEC	DEC Z HEXL	
021A	18		CLC	} ADD 1 TO THE DECIMAL NUMBER, USING X & Y AS TWO BYTE ACCUMULATOR
021B	98		TYA	
021C	69 01		ADC #01	
021E	A8		TAY	
021F	8A		TXA	
0220	69 00		ADC #00	
0222	AA		TAX	
0223	90 E9		BCC DECRHEX	
0225	E6 22		INC Z DECOUT	
0227	B0 E5		BCS DECRHEX	
0229	84 20	DEAD	STY Z HEXL	— FINISHED, SO STORE X & Y
022B	86 21		STX Z HEXH	
022D	A2 20		LDX #HEXL	
022F	20 64	FE	JSR QHEXTD1	— DISPLAY 4 DIGITS
0232	88		DEY	
0233	A5 22		LDA Z DECOUT	
0235	20 7A	FE	JSR HEXTD	— DISPLAY 5 DIGIT
0238	4C 04	FF	JMP RESTART	
023A				



A IS STORED IN 17 WHICH IS 10 THE "BASE ADDRESS" + 07 THE "INDEX"

IF WE DO

A2 07 LDX # 07

95 10 STA Z, X 10

THE STORE IS TO LOCATION 17 (=10 + X). THE ADDITION IS STRAIGHT-FORWARD BINARY, TRUNCATED TO A LOCATION IN ZERO PAGE SO

LDX # FF

STA Z, X 10

STORES IN LOCATION 0F

WE ALSO HAVE

STA, X 9D "STORE A INDEXED BY X"

STA, Y 99 "STORE A INDEXED BY Y"

(BUT NO STA Z, Y) WHICH DO NOT NEED TO TRUNCATE THE ADDITION THEY EXPECT A TWO BYTE ADDRESS SO

LDX # FF

STA, X 0010

STORES IN LOCATION 010F

NOW

DEX CA "DECREMENT (IN HEX) X BY ONE"

SETS THE Z FLAG IF X IS ZERO, & THE N FLAG EQUAL TO BIT 7 OF X.

BPL 10 "BRANCH IF PLUS"

TAKES THE BRANCH IF THE N FLAG IS CLEAR I.E. IS SAYING 'NOT NEGATIVE' I.E. PLUS. IT'S EASY TO SEE THAT THE COMBINATION



DECREMENTS X ONCE, AND, IF THE RESULT WAS POSITIVE (I.E. IN THE RANGE 0 – 7F) IT TAKES THE BRANCH AND DECREMENTS X AGAIN. ... AND AGAIN UNTIL IT REACHES A NON-POSITIVE NUMBER, WHICH WILL BE FF, WHEN IT DOESN'T TAKE THE BRANCH. IF WE START AT 7 AND EACH TIME AROUND THE LOOP CLEAR THE RELEVANT DISPLAY:

CODE	LABEL	MNEMONICS	COMMENT
A9 00		LDA #00	LOAD ACCUMULATOR IMMEDIATE
A2 07		LDX #07	LOAD X IMMEDIATE
95 10		LOOP: STA Z, X 10	STORE X IN ZERO PAGE INDEXED BY X
		DEX	DECREMENT X BY ONE
CA 10 FB		BPL LOOP	BRANCH IF PLUS TO "LOOP"

SO WE CAN WRITE A VERY SHORT PROGRAM TO CLEAR THE DISPLAY. BY MAKING THE LOOP SLIGHTLY LARGER (WITH THE SAME LENGTH OF PROGRAM)

```
0060 A2 07      LDX #07
0062 B5 48      LOOP:LDA Z, X 48
0064 95 10      STA Z,X 10
0066 CA         DEX
0067 10 F9      BPL LOOP
0069 4C 04 FF      JMP FF04
```

WE CAN, INSTEAD OF CLEARING THE DISPLAY, CAUSE A BLOCK OF MEMORY, 0048 – 004F, TO BE TRANSFERRED TO THE DISPLAY. THE PROGRAM IS POSITION INDEPENDENT SO YOU CAN WRITE IT INTO MEMORY ANYWHERE. . . EXCEPT LOCATIONS 0010 – 0017. IF YOU PUT THE PROGRAM IN 0048. . . IT WILL FUNCTION PERFECTLY BUT YOU WON'T BE ABLE TO CHANGE THE DATA WHICH IS MOVED, SINCE THIS IS THE PROGRAM. YOU CAN TRY THE PROGRAM USING THIS DATA

```
0048      00 77 58 5C 50 54 00 00
```

OR YOU COULD CONSTRUCT YOUR OWN DATA, USING APPENDIX A. THE INDEXING MECHANISM SHOWN ABOVE IS ONLY CAPABLE OF DEALING WITH 256 (CONSECUTIVE) BYTES, STARTING AT A GIVEN ADDRESS. THUS

```
A9 00      LDA #00      LOAD A IMMEDIATE WITH "00"
A8         TAY          TRANSFER A TO Y
18         LOOP:CLC      CLEAR CARRY
79 00 FE    ADC, Y FE00   ADD WITH CARRY INDEXED BY Y
C8         INY          INCREMENT Y
D0 F9      BNE LOOP      BRANCH IF NOT EQUAL
20 60 FE    JSR FE60      JUMP SUBROUTINE
4C 04 FF    JMP FF04      JUMP
```

COMPUTES THE LOWEST BYTE OF THE 256 BYTE ADDITION. (NOTE THAT, SINCE Y IS ZERO WHEN YOU LEAVE THE MONITOR BY THE GO FUNCTION, THE INITIALISATION OF A & Y CAN BE ACCOMPLISHED BY TYA INSTEAD OF LDA #00, TAY) HOW COULD THIS BE DONE FOR ALL 65536 MEMORY BYTES? CLEARLY IT IS POSSIBLE TO HAVE AN ADC, Y FOR EACH PAGE:

```
98         TYA
18         LOOP:CLC
79 00 00    ADC, Y 0000
18         CLC
79 00 FF    ADC, Y 0100
:          :
18         CLC
79 00 FF    ADC, Y FF00
C8         INY
F0 03      BEQ END
4C ??      JMP LOOP
20 60 FE    JSR FE60
4C 04 FF    JMP FF04
```

} 256 ADC, Y INSTRUCTION PAIRS

IN ORDER TO SHORTEN THIS PROGRAM WE WILL INTRODUCE THE CONCEPT OF "INDIRECTION".

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0240	A2 24		LDX #24	- SET UP X
0242	20 66 FE		JSR QHEXTD2	- PUT NEXT 4 OUT
0245	4C 04 FF		JMP RESTART	- DISPLAY RESULT
0247				

SYSTEM

THE DECIMAL TO HEX CONVERTER WILL PROMPT WITH 0XXXX FOR THE FIRST DIGIT OF THE 5 DIGIT DECIMAL NUMBER. THEN X0000. FOR THE LAST FOUR DIGITS OF THE DECIMAL NUMBER. CLEARLY ANYTHING OVER 65535 WILL GIVE THE REMAINDER WHEN DIVIDED BY 10000 HEX. TO ENTER THIS NUMBER YOU WOULD KEY 6, CONTROL KEY, 5535, CONTROL KEY, AND FFFF WILL APPEAR ON THE DISPLAY (AFTER A SLIGHT DELAY!) THE PROGRAM WORKS BY A PROCESS OF DECREMENTING THE DECIMAL NUMBER AND THEN INCREMENTING THE HEX. NUMBER.

DEC→HEX

0200	98		TYA	- CLEAR A
0201	85 20		STA Z DECL	- CLEAR NO
0203	85 21		STA Z DECH	
0205	A2 20		LDX #DECC	
0207	85 22	AGAIN	STA Z DECVH	} - FETCH THE FIRST DIGIT
0209	20 7A FE		JSR HEXTD	
020C	20 0C FE		JSR DISPLAY	
020F	90 F6		BCC AGAIN	
0211	20 88 FE		JSR QDATFET	- AND THEN THE LAST FOUR DIGITS
0214	F8		SED	- DECIMAL MODE
0215	84 10		STY Z D	- CLEAR LEFT DISPLAY
0217	A6 21		LDX Z DECH	- X & Y AS DOUBLE ACCUMULATOR
0219	98		TYA	
021A	85 21		STA Z DECH	- CLEAR AREA FOR RESULT
021C	A4 20		LDY Z DECL	
021E	85 20		STA Z DECL	
0220	38	NEXT	SEC	} - DO A DECIMAL SUBTRACT, DOUBLE BYTE
0221	98	ALSO	TYA	
0222	E9 01		SBC #01	
0224	A8		TAY	
0225	8A		TXA	
0226	E9 00		SBC #00	
0228	AA		TAX	
0229	B0 04		BCS NODEC	
022B	C6 22		DEC Z DECVH	- LAST OF THE DECIMAL SUBTRACT, TO DO 5 DIGITS
022D	30 09		BMI RESULT	- IF MINUS THEN FINISHED
022F	E6 20	NODEC	INC Z DECL	- DOUBLE HEX INCREMENT
0231	D0 ED		BNE NEXT	
0233	E6 21		INC Z DECH	
0235	38		SEC	} - CREATE BRANCH ALWAYS, BUT DON'T BOTHER TO SET THE CARRY TWICE
0236	B0 E9		BCS ALSO	
0238	A2 20	RESULT	LDX #20	
023A	20 64 FE		JSR QHEXTD	- DISPLAY RESULT
023D	4C 04 FF		JMP RESTART	
023F				

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0208	A2 20		LDX #20	
020A	20 88 FE		JSR QDATFET	– FETCH THE NUMBERS
020D	98		TYA	– CLEARS A
020E	A0 08		LDY #08	– LOOP COUNTER
0210	66 20	LOOP	ROR Z 20	– SHIFT MULTIPLIER (AND HIGH BYTE OF RESULT)
0212	90 03		BCC NAD	– NO ADD IF NO BIT
0214	18		CLC	
0215	65 21		ADC Z 21	– ADD MULTIPLICAND INTO LOW BYTE OF RESULT
0217	6A	NAD	ROR A	– AND SHIFT LOW BYTE OF RESULT
0218	88		DEY	
0219	D0 F5		BNE LOOP	
0213	85 21		STA Z 21	– PUT IN LOW BYTE
021D	66 20		ROR Z 20	– FINAL JUSTIFICATION SHIFT
021F	20 64 FE		JSR QHEXTD	– DISPLAY ANSWER
0222	20 64 FF		JMP RESTART	
0224				

DOUBLE BYTE MULTIPLY

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0200	D8		CLD	– BINARY ONLY
0201	84 20	}	STY Z 20 MPIER	– FORM PROMPT FOR THE ZERO INPUT
0203	84 21		STY Z 21	
0205	A9 11		LDA #11	
0207	85 22	}	STA Z 22 MPICAND	– FORM PROMPT FOR THE FIRST INPUT
0209	85 23		STA Z 23	
020B	A2 20		LDX #20	
020D	20 88 FE		JSR QDATFET	– FETCH ZERO INPUT
0210	A2 22		LDX #22	
0212	20 88 FE		JSR QDATFET	– AND FIRST INPUT
0215	84 24		STY Z 24	– CLEAR WORKING SPACE
0277	84 25		STY Z 25	
0219	A0 10		LDY #10	– LOOP COUNT INITIALISATION
021B	66 23	LOOP	ROR Z 23	– TWO BYTE SHIFT RIGHT
021D	66 22		ROR Z 22	
021F	90 0D		BCC NAD	– NO ADD IF THE O/P BIT ISN'T A ONE
0221	18		CLC	
0222	A5 20		LDA Z 20	– TWO BYTE ADD
0224	65 24		ADC Z 24	
0226	85 24		STA Z 24	
0228	A5 21		LDA Z 21	
022A	65 25		ADC Z 25	
022C	85 25		STA Z 25	– NO CARRY OUT OF THE ADD
022E	66 25	NAD	ROR Z 25	– SHIFT AGAIN
0230	66 24		ROR Z 24	
0232	88		DEY	
0233	D0 E6		BNE LOOP	– GO ROUND LOOP 16 TIMES
0235	66 23		ROR Z 23	– FINAL SHIFT ON RESULT
0237	66 22		ROR Z 22	
0239	A0 06		LDY #06	– SET UP POSITION
023B	20 66 FE	FE	JSR QHEXTD2	– X ALREADY POINTING AT CORRECT LOCATIONS – PUT 4 HEX OUT
023E	A0 02		LDY #02	– NEXT POSITION

4.3.INDIRECTION:

YOU'LL NOTICE THAT THE PROGRAM IS NOT POSITION INDEPENDENT: THE ADDRESS OF THE CLC INSTRUCTION MUST BE WRITTEN INTO THE PROGRAM. THIS IS ANOTHER DISADVANTAGE OF THIS METHOD: (THERE IS AN ADVANTAGE: THIS PROGRAM IS VERY FAST, TAKING ONLY 6μS PER BYTE). THE INSTRUCTION REQUIRED MUST HAVE A 16 BIT UNFIXED ADDRESS AND THIS CAN ONLY GO IN ONE PLACE : MEMORY. A LIMITATION IS THAT GENERALLY IT CAN ONLY BE IN ZERO PAGE MEMORY. THE CONCEPT IS KNOWN AS INDIRECTION. THE MOST DIRECT VERSION OF THIS IS THE INDIRECT JUMP.

6C 02 00 JMP (0002)

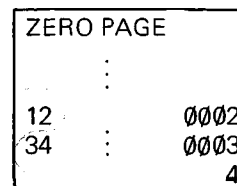
THIS IS THE ONE VERSION OF INDIRECTION THAT DOESN'T NEED TO REFER TO ZERO PAGE MEMORY. WHAT HAPPENS IS THIS:

TIME, μS	ADDRESS BUS	DATA BUS	R/W
0	PC	6C	1 JUMP INDIRECT
1	PC+1	02	1
2	PC+2	00	1
3	0002	V	1 LOWER BYTE
4	0003	U	1 HIGHER BYTE
5	UV	OPCODE	1 OLD 6C COMPLETED

THE MONITOR USES A JUMP INDIRECT FOR THE GO FUNCTION, HAVING BUILT THE ADDRESS IN 0002 & 0003 : A JUMP INDIRECT VIA 0002 & 0003, ASSUMING THAT THESE LOCATIONS HAVEN'T BEEN ALTERED, WILL THUS RETURN TO THE START OF THE PROGRAM – WITHOUT KNOWING WHERE IT HAD BEEN ENTERED INTO MEMORY AT THE TIME OF WRITING.

INDIRECT JUMP

MAIN PROGRAM



ROUTINE

XX	1234
XX	1235
XX	1236
" "	" "

WELL, THAT WAS SIMPLE INDIRECTION. NOW WE'LL MOVE ONTO THE MORE COMPLICATED MODES OF INDIRECTION. HAVING FETCHED THE ADDRESS OUT OF MEMORY WITH THE INDIRECTION STAGE, WE CAN INDEX IT. THIS IS CALLED POST-INDEXED INDIRECTION. WITH THE 65XX SERIES OF MICRO-PROCESSORS YOU MAY ONLY

- I INDEX IN THIS MODE WITH THE Y INDEX REGISTER
- II USE ZERO PAGE MEMORY

TIME, μ S	ADDRESS BUS	DATA BUS	R/W	
0	PC	B1	1	LDA (I),Y
1	PC+1	I	1	
2	00I	J	1	
3	00I+1	K	1	
4	KJ+Y	DATA	1	(AN EXTRA μ S IS NEEDED IF J+Y
5	PC+2	OPCODE	1	RESULTS IN A CARRY)
THIS IS THE MODE OF ADDRESSING NEEDED TO SOLVE THE 65536 BYTE ADDITION PROBLEM. MEANWHILE WHAT ABOUT THE X REGISTER AND INDIRECTNESS? HERE WE HAVE PRE-INDEXED INDIRECTION				
TIME, μ S	ADDRESS BUS	DATA BUS	R/W	
0	PC	A1	1	LDA (I,X)
1	PC+1	I	1	
2	00I	DATA,	1	
		DISCARDED		
3	00I+X	J	1	NO CARRY TO HIGH ORDER BYTE
4	00I+X+1	K	1	
5	KJ	DATA	1	PUT IN A
6	PC+2	OP CODE	1	

THIS IS THE OPPOSITE TO POST-INDEXED ... HERE THE INDEXING SWITCHES BETWEEN DIFFERENT INDIRECTION LOCATIONS. THE EFFECTS OF THESE TWO INDEXING MODES ARE ONLY THE SAME IN THE TRIVIAL CASE OF ZERO INDEXES. HERE IS THE SOLUTION TO THE 65536 BYTE ADDITION:

98		TYA		-ZERO Y & A
85 20		STA Z 20	}	SET UP INDIRECT LOCATIONS
85 21		STA Z 21		
18	LOOP	CLC		
71 20		ADC (20), Y		
C8		INY		
D0 FA		BNE LOOP		
E6 21		INC Z 21		
D0 F6		BNE LOOP		
20 60 FE		JSR FE60		
4C 04 FF		JMP FF04		

THE PROGRAM IS, ONCE AGAIN, POSITION INDEPENDENT. IT IS, AS IMPLIED IN THE FIRST SOLUTION, SLOW : 12 μ S PER BYTE. THIS IS MAINLY DUE TO THE SMALL SIZE OF THE LOOP : THE 3 μ S 'NEARLY ALWAYS TAKEN' BRANCH IS TAKING A DISPROPORTIONATE AMOUNT OF TIME, IN THE FIRST SOLUTION THE EQUIVALENT 5 μ S BRANCH AND JUMP COMBINATION OCCURS ONLY EVERY 256 BYTES AND IS THUS IGNORED IN THE TIME CALCULATIONS. THE INSTRUCTION INC Z 21 HAS AN OBVIOUS FUNCTION : INCREMENT (IN HEXADECIMAL) LOCATION 0021. IT ACTS JUST LIKE INX OR INY - BUT IT TAKES 5 μ S INSTEAD OF 2 μ S.

4.4 READ-MODIFY WRITE INSTRUCTIONS

THERE ARE COMPANION INSTRUCTIONS TO INC Z THAT CAN DIRECTLY ALTER MEMORY CONTENTS, THESE ARE CALLED READ-MODIFY-WRITE INSTRUCTIONS, THE NEXT OF WHICH IS THE OBVIOUS DEC INSTRUCTION.

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0217	A4 20		LDY Z 20	- USE Y & X AS DOUBLE ACCUMULATOR
0219	A6 21		LDX Z 21	
021B	38		SUB SEC	
021C	98		TYA	
021D	E5 22		SBC Z 22	- SUBTRACT THE DIVISOR
021F	A8		TAY	
0220	8A		TXA	
0221	E9 00		SBC #00	
0223	AA		TAX	
0224	90 10		BCC RESULT	- IF NEGATIVE THEN FINISHED
0226	84 23		STY Z 23	- ELSE UPDATE THE REMAINDER
0228	A5 24		LDA Z 24	
022A	69 00		ADC #00	
022C	85 24		STA Z 24	- AND ADD ONE TO THE RESULT (CARRY WAS SET ON INPUT).
022E	A5 25		LDA Z 25	
0230	69 00		ADC #00	
0232	85 25		STA Z 25	
0234	90 E5		BCC SUB	- NO CARRY IS POSSIBLE (USUALLY)
0236	A2 24	RESULT	LDX #24	
0238	20 64 FE		JSR QHEXTDI	- DISPLAY RESULT
023B	A5 23		LDA Z 23	
023D	20 60 FE		JSR RDHEXTD	- AND REMAINDER
0240	4C 04 FF		JMP RESTART	
0242				

THE TWO MULTIPLY ROUTINES ARE FOR SINGLE AND DOUBLE BYTE BINARY MULTIPLICATION. THE FIRST PROMPTS XX0011XX AND THE TWO NUMBERS TO BE MULTIPLIED SHOULD BE ENTERED SEQUENTIALLY. (E.G. 1234 WOULD GIVE 12 X 34). THE SECOND PROMPTS XX0000XX FOLLOWED BY XX1111XX FOR THE TWO NUMBERS. ANSWERS ARE, AS USUAL, DISPLAYED AFTER A CONTROL KEY HAS BEEN PRESSED.

BOTH ARE BASED ON AN EQUIVALENT TO THE NORMAL METHOD OF LONG MULTIPLICATION.

E.G.

11010	
00110	
0000000000	- (0 X 2 ⁴) X 11010
0000000000	- (0 X 2 ³) X 11010
1101000	- (1 X 2 ²) X 11010
110100	- (1 X 2) X 11010
000000	- (0 X 2 ⁰) X 11010
10011100	

SINGLE BYTE MULTIPLY

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0200	D8		CLD	
0202	84 20		STY Z 20	- SET UP PROMPT FOR ZERO - MULTIPLIER
0204	A9 11		LDA #11	
0206	85 21		STA Z 21	- PROMPT FOR FIRST - MULTIPLICAND

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
020F	84 23		STY Z SUBL	- SUBTRACT 0001 AT START
0211	A4 20		LDY Z SQL	- USE Y & X AS DOUBLE SIZED ACCUMULATOR
0213	A6 21		LDX Z SQH	
0215	38	NXTSUB	SEC	
0216	98		TYA	
0217	E5 23		SBC Z SUBL	
0219	A8		TAY	
021A	8A		TXA	
021B	E5 24		SBC Z SUBH	
021D	AA		TAX	
021E	90 14		BCC RESULT	- IF NEGATIVE THEN STOP
0220	A9 00		LDA # 00	- NOT FINISHED YET. INCREMENT ROOT
0222	65 22		ADC Z ROOT	
0224	85 22		STA Z ROOT	
0226	A5 23		LDA Z SUBL	- INCREMENT SUB
0228	69 02		ADC #02	
022A	85 23		STA Z SUBL	
022C	A5 24		LDA Z SUBH	
022E	69 00		ADC #00	
0230	85 24		STA Z SUBH	
0232	90 E1		BCC NXTSUB	- THERE CAN BE NO CARRY: BRANCH ALWAYS
0234	A5 22	RESULT	LDA Z ROOT	
0236	20 60 FE		JSR DHEXTD	- DISPLAY ANSWER
0239	4C 04 FF		JMP RESTART	
024B				

THE DIVIDE ROUTINE WILL CALCULATE THE INTEGER RESULT AND REMAINDER OF A FOUR DIGIT NUMBER DIVIDED BY A TWO DIGIT NUMBER. BY ENTERING CLD (FOR HEX.) OR SED (FOR DECIMAL) EITHER BASE MAY BE USED, SINCE IT WORKS BY SUBTRACTING THE DIVISOR SUCCESSIVELY FROM THE DIVIDEND. THE ROUTINE PROMPTS WITH XX0000XX FOR THE DIVIDEND AND THEN XXXX11XX FOR THE DIVISOR. THE ANSWER WILL APPEAR IN THE FORM ABCD.EF WHERE ABCD IS THE INTEGER RESULT AND EF IS THE REMAINDER.

DIVIDER

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
0200	D8 OR	F8	CLD OR SED	- BINARY (DECIMAL) OPERATION
0201	84 20		STY Z 20 DIVIDED	- CLEAR DIVIDEND - PROMPT FOR NUMBER
0203	84 21		STY Z 21	
0205	A9 11		LDA #11	- PROMPT FOR SECOND NUMBER
0207	85 22		STA Z 22 DIVISOR	
0209	A2 20		LDX #20	
020B	20 88 FE		JSR QDATFET	- FETCH DIVIDEND
020E	A2 22		LDX #22	
0210	20 88 FE		JSR QDATFET	- FETCH DIVISOR
0213	84 24		STY Z 24 RESULT	- CLEAR RESULT
0215	84 25		STY Z 25	

THE OTHER FOUR ARE NEW, THEY ARE SHIFTS AND ROTATES. LET'S USE ASL AS AN EXAMPLE

0070	A9 55	LDA #55	LOAD A IMMEDIATE WITH 55
72	0A	ASLA	ARITHMETIC SHIFT LEFT
73	20 60 FE	JSR FE60	JUMP TO SUBROUTINE
76	4C 04 FF	JMP FF04	JUMP

THE RESULT OF RUNNING THIS PROGRAM IS AA ON THE DISPLAY. EACH BIT IN THE ACCUMULATOR HAS BEEN SHIFTED ONE BIT LEFT.

BEFORE
AFTER

ROLA, ROTATE LEFT ACCUMULATOR, (2A) WILL HAVE THE SAME EFFECT, EXCEPT THAT THE RIGHT INPUT 0 IS REPLACED BY C, IN THIS CASE 1, SO THE RESULT IS AB.

LSRA, LOGICAL SHIFT RIGHT ACCUMULATOR (4A)

BEFORE
AFTER

RORA, ROTATE RIGHT ACCUMULATOR (6A) WILL REPLACE THE LEFT INPUT 0 WITH C TO GIVE AA

ALL THESE INSTRUCTIONS MAY BE USED DIRECTLY ON MEMORY LIKE INC Z.

4.5 MISCELLANEOUS REMAINING INSTRUCTIONS

THERE ARE A FEW INSTRUCTIONS LEFT, WHICH WILL HAVE TO BE DEALT WITH PIECE-MEAL:

BRK 00 : THE MICROPROCESSOR HAS TWO INTERRUPTS, AS EXPLAINED IN THE HARDWARE SECTION, AND THE INSTRUCTION SIMULATES AN IRQ, FIRST SETTING THE B FLAG IN THE STATUS REGISTER. THE RETURN AFTER A BREAK WILL BE AT THE NEXT BUT ONE BYTE

BIT 2C : A COMBINATION OF TWO INSTRUCTIONS

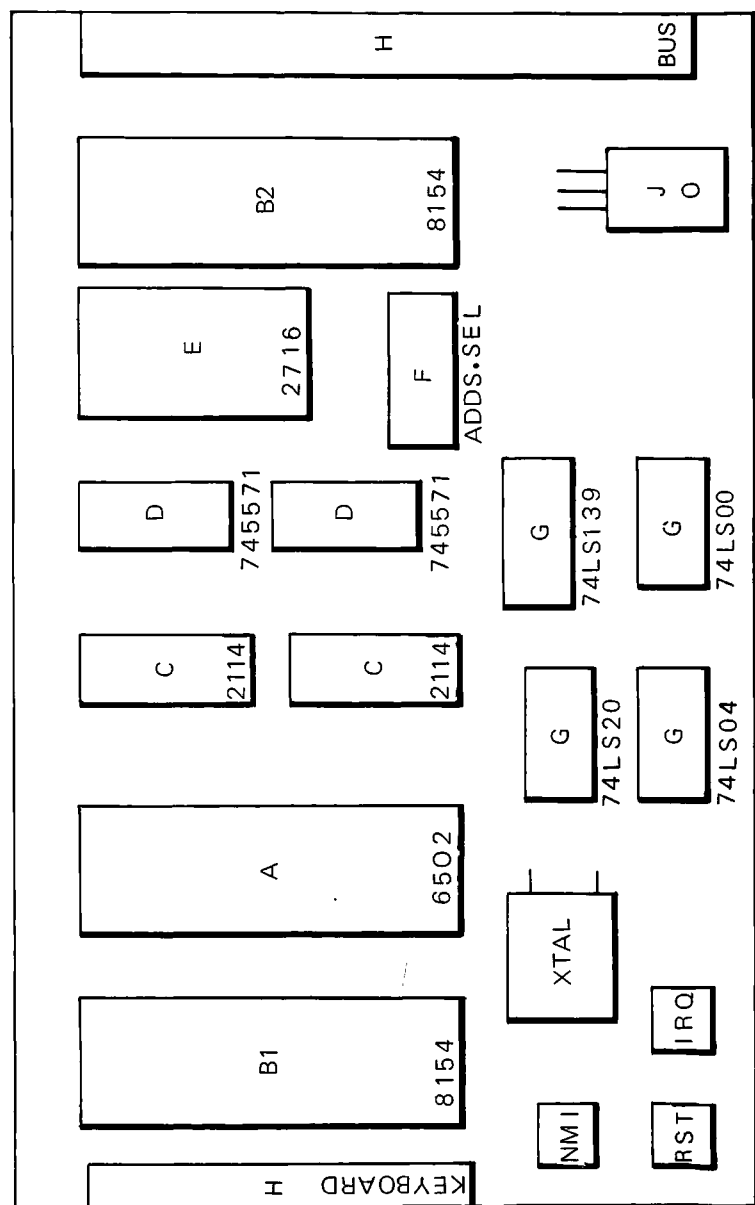
- I READ MEMORY BITS 6 & 7 INTO THE OVERFLOW & NEGATIVE FLAGS
- II LOGICAL AND ACCUMULATOR AND MEMORY, A ZERO RESULT SETTING THE Z FLAG. THE RESULT IS NOT LOADED INTO THE ACCUMULATOR. THE INSTRUCTION IS USUALLY USED TO TEST THE STATUS OF PERIPHERAL DEVICES, WITHOUT UPSETTING A, X OR Y.

RTI, RTS 40, 60 BOTH INSTRUCTIONS PULL THE PROGRAM COUNTER FROM THE STACK, RTI FIRST PULLS THE PROCESSOR STATUS FROM THE STACK.

CHAPTER 5: ACORN HARDWARE

5.1 CHIP LAYOUT AND BUS

BEFORE PLUNGING DEEPER INTO SOFTWARE WE'LL TAKE A REST AND LOOK AT THE HARDWARE. WE'LL START WITH THE CPU BOARD



SECTION 5.1

ROUTINES BY ENTERING THE SECTION OF PROGRAM FROM THE TITLE LABEL (E.G. DIVIDE) TO THE RESULT LABEL AND SUBSTITUTING THE LINE

60 RESULT RTS .
ALL ARE RELOCATABLE.

SYSTEM PROGRAMS

THESE PROGRAMS ARE ALL SHORT ROUTINES WHICH CAN PROVE USEFUL TIME SAVERS AT THE DEVELOPMENT AND INPUT STAGES OF PROGRAM WRITING.

IT MAY BE FOUND USEFUL TO KEEP COPIES OF THEM ON TAPE AND TO HAVE THEM IN THE ACORN AND BESIDE YOU WHILE DEVELOPING PROGRAMS. BRANCH CALCULATIONS IN PARTICULAR ARE A FERTILE SOURCE OF ERRORS AND TIME WASTING IN ANY HAND ASSEMBLED PROGRAM. THE RELOCATOR WILL MOVE PROGRAMS AROUND MEMORY FOR YOU. A GODSEND TO ANYONE WHO FINDS THEMSELVES WITH THE NEED TO REENTER LARGE CHUNKS OF CODE MANUALLY.

MISCELLANEOUS

THIS IS A SELECTION OF PROGRAMS AND ROUTINES INCLUDED BECAUSE THEY ARE INTERESTING, ELEGANT OR IMPORTANT. THEY SHOW SOME OF OF THE THINGS THAT CAN BE DONE WITH THE SYSTEM, WHICH MAY BE MORE THAN YOU IMAGINE. WE HAVE, FOR INSTANCE, RUN A CHESS GAME IN THE 1K SYSTEM.

IN PARTICULAR THE METRONOME AND COUNTER PROGRAMS ARE INTENDED TO DEMONSTRATE SOME OF THE USES OF THE KEYBOARD. IN ORDER TO UNDERSTAND WHAT IS GOING ON WITH THESE PROGRAMS YOU WOULD BE WELL ADVISED TO STUDY THE MONITOR LISTING AND PART 1 OF THIS MANUAL.

MATHEMATICAL

THE SQUARE ROOT PROGRAM WILL CALCULATE EITHER DECIMAL OR HEXA-DECIMAL SQUARE ROOTS ACCORDING AS CLD (FOR HEX) OR SED (FOR DECIMAL) IS ENTERED AS THE FIRST LINE. IN EITHER CASE THE PROMPT WILL BE `XX0000XX`. THE SQUARE SHOULD BE ENTERED, A CONTROL KEY PRESSED AND THE ROOT WILL APPEAR ON THE DISPLAY.

THE PROGRAM IS BASED ON THE EQUALITY

$$((N+1)*(N+1))-(N*N)=(2*N)+1$$

AND SUCCESSIVELY SUBTRACTS 1,3,7,9 ETC. FROM THE SQUARE. WHEN THE RESULT OF A SUBTRACTION GOES NEGATIVE THE NUMBER OF SUBTRACTIONS DONE TO DATE IS THE ROOT.

HEX/DEC SQ ROOT.

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS	RELOCATABLE
0200	F8	OR D8	SED (OR CLD)	- SET DECIMAL (BINARY) OPERATING	
0201	84	21	STY Z SQH	- CLEAR SQUARE TO PROMPT	
0203	84	20	STY Z SQL		
0205	A2	20	LDX #SQL		
0207	20	88 FE	JSR QDATFET	- FETCH THE NO. WHOSE ROOT IS TO BE FOUND	
020A	84	24	STY Z SUBH	- CLEAR HIGH PART OF SUBTRACTING NO.	
020C	84	22	STY Z ROOT	- CLEAR ROOT	
020E	C8		INY		

PART 2 APPLICATION PROGRAMS

MATHEMATICAL

1. SQUARE ROOT (HEX. OR DECIMAL)
2. DIVIDE (HEX. OR DECIMAL)
3. SINGLE BYTE MULTIPLY
4. DOUBLE BYTE MULTIPLY

SYSTEM

1. DECIMAL TO HEX.
2. HEX. TO DECIMAL
3. BRANCH OFFSET CALCULATOR
4. RELOCATOR
5. TAPE USE PROGRAMS
6. SCROLL

GAMES

1. NIM
2. DUCK SHOOT

MISCELLANEOUS

1. COUNTER
2. KEYBOARD COUNTER ROUTINE
3. METRONOME
4. EIGHT QUEENS PROBLEM

GENERAL

THESE APPLICATIONS PROGRAMS ARE INTENDED TO DEMONSTRATE SOME OF THE CAPABILITIES OF THE SYSTEM AND OF THE PROCESSOR. THEY HAVE BEEN DESIGNED FOR CLARITY AND SIMPLICITY AND IN MANY CASES ARE NOT OPTIMAL EITHER IN TERMS OF LENGTH OF PROGRAM OR OF EXECUTION TIME. THEY ARE INTENDED SIMPLY TO GIVE YOU A FEEL FOR THE SYSTEM AND SOMEWHERE TO START OFF FROM.

ALL PROGRAMS MARKED RELOCATABLE CAN BE ENTERED ANYWHERE IN AVAILABLE MEMORY, SUBJECT TO NOT IMPINGING IN VARIABLE STORAGE SPACE FOR EITHER THE PROGRAM OR MONITOR AND NOT USING SPACE NEEDED BY THE STACK. (FOR STACK USAGE SEE RELEVANT SECTIONS OF PART 1 OF THIS MANUAL.)

AS FAR AS HAS PROVED POSSIBLE THE CONVENTION OF A XX 0000 XX PROMPT FOR THE FIRST NUMBER TO BE ENTERED AND XX 1111 XX FOR THE SECOND HAS BEEN OBSERVED IN THESE PROGRAMS. AFTER ENTERING A NUMBER CHECK THAT IT IS CORRECT AND THEN PRESS A CONTROL KEY (ANY ONE WILL DO) TO PROGRESS THROUGH THE PROGRAM.

YOU SHOULD NOW BE READY TO TYPE IN THE PROGRAMS AND RUN THEM, BOTH TO ASSURE YOURSELF THAT THE SYSTEM IS OPERABLE AND TO FAMILIARISE YOURSELF WITH ITS OPERATION.

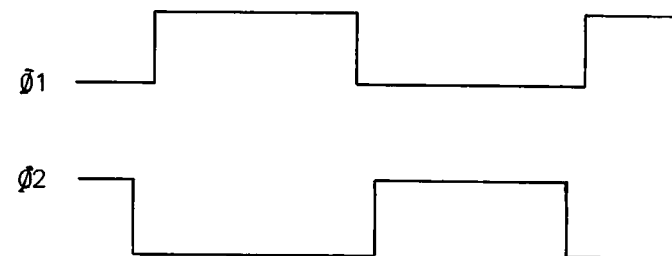
THROUGHOUT THESE NOTES X INDICATES AN UNDEFINED/UNIMPORTANT CHARACTER.

MOST OF THE PROGRAMS WERE WRITTEN BY MARK I'ANSON, THANK YOU MARK I.

MATHEMATICAL PROGRAMS

ALL THESE ROUTINES RESET THEMSELVES WHEN A CONTROL KEY IS PRESSED AFTER THE NUMBER HAS BEEN OBTAINED, THEY MAY ALL BE USED AS SUB

THE OBVIOUS IMPORTANT DEVICE HERE IS A, THE MICROPROCESSOR. THIS IS WHERE A,X,Y,P,S,PC LIVE. FROM HERE COME THE COMMANDS TO RUN EVERYTHING ELSE. THERE ARE TWO PRIMARY BUSES, CONSISTING OF PARALLEL PATHS OF BINARY DATA, THE BIGGEST BUS IS THE ADDRESS BUS. THIS CONSISTS OF 16 LINES TO TRANSFER THE ADDRESS GENERATED BY THE PROCESSOR TO THE ADDRESS INPUTS OF ALL OTHER SYSTEM CHIPS. THIS BUS IS UNIDIRECTIONAL : ONLY THE PROCESSOR (IN A NORMAL SYSTEM) GENERATES ADDRESSES, AND IT HAS 2^{16} STATES (=65536,) THE SECOND BUS IS THE DATA BUS. THIS IS 8 BI-DIRECTIONAL LINES, ALLOWING A SINGLE WORD/BYTE TO BE TRANSFERRED EITHER FROM THE PROCESSOR TO MEMORY – A WRITE, OR FROM MEMORY TO PROCESSOR – A READ. THE REMAINING BUS IS THE CONTROL BUS, ITS MEMBERS HAVE NO PARTICULAR RELATIONSHIP WITH EACH OTHER, BUT THEY ARE ALL SUPERVISORY SIGNALS FOR THE SYSTEM. THE FIRST CONTROL SIGNAL IS THE R/W LINE. THIS SPECIFIES THE TYPE OF DATA TRANSFER THAT THE PROCESSOR WISHES TO MAKE: WHEN THE R/W LINE IS HIGH (LOGIC ONE; > 2.4 V DC) THE PROCESSOR IS READING WHEN THE R/W LINE IS LOW (LOGIC ZERO < 0.8 V DC) THE PROCESSOR IS WRITING, THE NEXT CONTROL LINES ARE THE SYSTEM CLOCK, WHICH CONTROLS THE TIMING OF ALL DATA TRANSFERS. THE PROCESSOR, WITH HELP FROM 1/6 OF A TTL IC, GENERATES THE SYTEM CLOCK AS TWO NON-OVERLAPPING SQUARE WAVES, PHASE ONE (01) & PHASE TWO (02)



DURING 01 THE ADDRESS BUS AND THE R/W LINE CHANGE, AT THE END OF, OR DURING, 02 THE DATA IS TRANSFERRED. OTHER CONTROL SIGNALS ALSO CHANGE AT TIMES SPECIFIED WITH RESPECT TO THE SYSTEM CLOCK, E.G. THE SYNC SIGNAL : THIS GOES HIGH DURING 01 WHEN THE PROCESSOR IS FETCHING AN INSTRUCTION, AND RETURNS LOW WITH THE TRAILING EDGE OF 02.

5.2 RESET. INTERRUPT REQUEST AND NON-MASKABLE INTERRUPT

ANOTHER CONTROL LINE IS RESET. THIS IS GENERATED BY SUITABLE HARDWARE (IN THE ACORN THE CORNER SWITCH ON THE CPU BOARD, AND THE RESET SWITCH ON THE KEYBOARD,) AND CAUSES ALL PARTS OF THE SYSTEM TO BE RESET TO A SAFE, KNOWN STATE. IN THE PROCESSOR'S CASE RESET INITIALIZES THE PROGRAM COUNTER TO THE CONTENTS OF ADDRESSES FFFC AND FFFD WHICH, FOR ACORN, CONTAIN THE ADDRESS FEF3. EXECUTION OF THE ACORN MONITOR STARTS THERE. PERIPHERAL DEVICES SHOULD BE SET TO THEIR LEAST DANGEROUS STATE BY RESET, E.G. REMOVE INTERRUPT CAPABILITY, SET ALL PROGRAMMABLE INPUT/OUTPUT LINES TO INPUTS.

THE TWO PUSH BUTTONS ON THE CPU BOARD ON EITHER SIDE OF THE RESET BUTTON ARE INTERRUPT BUTTONS. THE IDEA OF AN INTERRUPT IS TO PULL THE PROCESSOR AWAY FROM IT'S CURRENT TASK, LET IT BRIEFLY DO SOMETHING IMPORTANT AND THEN RETURN TO IT'S TASK AS IF NOTHING HAD HAPPENED. THE 6502 HAS TWO DISTINCT INTERRUPT CAPABILITIES

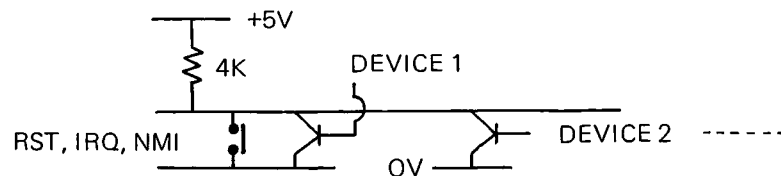
IRQ

WITH AN INTERRUPT REQUEST, IRQ, THE PROCESSOR HAS THE OPTION OF IGNORING IT. AN IRQ IS ONLY GRANTED IF THE FLAG I (INTERRUPT DISABLE) IN THE PROCESSOR STATUS REGISTER IS 0. THE PROCESSOR THEN PUSHES PC & P & THEN SETS I TO 1. (THE STATE OF THE IRQ LINE IS CHECKED BETWEEN INSTRUCTIONS . . . IF IT REMAINS LOW, WE DON'T WANT ANOTHER INTERRUPT). THEN THE PROCESSOR LOADS PC FROM LOCATIONS FFFE & FFFF AND CONTINUES. NOTE THAT AN RTI RETURNS THE ORIGINAL P, WHICH HAD THE I FLAG 0.

NMI

WITH A NON-MASKABLE INTERRUPT, NMI, THE PROCESSOR HAS NO OPTIONS; WHEN THE LINE HAS BEEN LOW FOR AT LEAST TWO CLOCK CYCLES, THE PROCESSOR WILL FINISH ITS CURRENT INSTRUCTION, SAVE ITS STATUS & PC, SET I HIGH AND FETCH A NEW PC FROM FFFA & FFFB. TO AVOID RECOGNISING ANOTHER INTERRUPT NMI IS EDGE-SENSITIVE: NO FURTHER INTERRUPTS ARE RECOGNISED UNTIL NMI HAS RETURNED HIGH. SINCE NMI SETS I HIGH, IRQ WILL NOT SUCCEED DURING THE NORMAL OPERATION OF AN NMI PROGRAM, BUT NMI WILL BE ABLE TO TAKE CONTROL DURING EXECUTION OF AN IRQ PROGRAM; IT HAS A HIGHER PRIORITY.

IRQ, NMI, & RESET ARE OPEN-COLLECTOR LINES ON THE CPU BOARD: MANY INTERRUPTING/RESETTING DEVICES MAY BE CONNECTED.

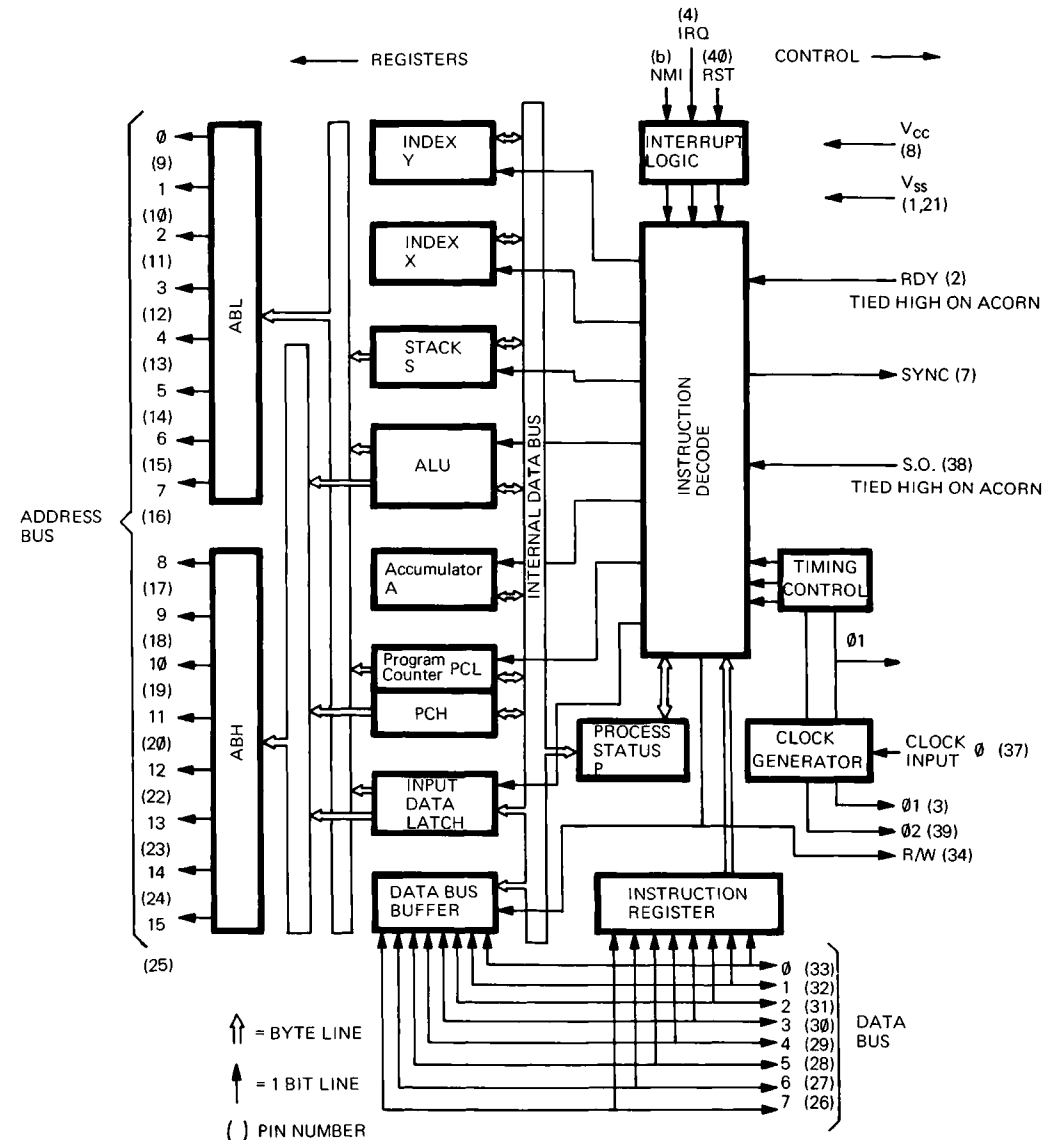


FF B3	85	0A		BREAK	STA Z R0	– WHEN THE IRQ/BREAK VECTOR POINTS HERE THEN DISPLAY EVERYTHING – SAVE A
B5	86	0B			STX Z R1	– SAVE X
B7	84	0C			STY Z R2	– SAVE Y
B9	68				PLA	– GET P OFF STACK
BA	48				PHA	– PUT IT BACK FOR FUTURE USE
BB	85	0D			STA Z R3	– STORE Q IN REGISTER 3
BD	A2	0D			LDX #R3	– SET X TO POINT AT REGISTERS 3 → 0 FOR QUAD
BF	A9	FF			LDA # FF	– KILL POSSIBILITY OF DISPLAY BEING ON SINGLE SCAN
C1	85	0E			STA Z REPEAT	
C3	20	00	FE		JSR QUAD	– USE QUAD TO WRITE OUT A X Y P
C6	BA				TSX	– GET STACK POINTER
C7	86	13			STX Z R7	
C9	C8				INY	– Y ZERO SINE QUAD ENDED WITH DISPLAY SO THIS FORMS 01
CA	84	12			STY Z R6	
CC	D8				CLD	– CLEAR DECIMAL MODE FOR BINARY SUBTRACT – DOESN'T AFFECT USER SINCE P IS STACKED
CD	BD	02	01		LDA, X 0102	– GET PCL OFF STACK
D0	38				SEC	
D1	E5	1F	0C		SBC Z RECAL	– CORRECT IT BY AMOUNT IN RECAL
D3	9D	02	01		STA, X 0102	– PUT IT BACK ON STACK
D6	85	11			STA Z R5	– AND STORE IT FOR QUAD
D8	BD	03	01		LDA, X 0103	– PCH OFF STACK
DB	E9	00			SBC #00	– REST OF TWO BYTE SUBTRACTION
DD	9D	03	01		STA, X 0103	– PUT IT BACK ON STACK
E0	85	10			STA Z R4	– AND STORE IT FOR QUAD
E2	A2	13			LDX #R7	– POINT X AT THESE REGISTERS – QUAD WILL DESTROY THEM
E4	20	00	FE		JSR QUAD	– QUAD WRITES OUT PC SP
FF E7	4C	07	FF		JMP RE-ENTER	– AND THE WHOLE SHEBARG STARTS OVER AGAIN
FF EA	3F	06	5B	4F	FONT	'0' '1' '2' '3' – 7 SEGMENT FORMS OF THE HEX DIGITS
EE	66	6D	7D	07		'4' '5' '6' '7'
F2	7F	6F	77	7C		'8' '9' 'A' 'b'
F6	58	5E	79	71		'c' 'd' 'E' 'F'
FF FA	AD	FF			NMIVEC	NMI – POINT TO THE ADDED INDIRECTION
FF FC	F3	FE			RSTVEC	RESET – POINT TO THE RESET ENTRY POINT
FF FE	B0	FF			IRQVEC	IRQ – POINT TO THE ADDED INDIRECTION

FF45	20	5E	FE	"MODIFY"	JSR MHEXTD	- DISPLAY THE MEMORY
48	20	0C	FE		JSR DISPLAY	- AND GET KEY
4B	B0	BC			BCS SEARCH	- IF NOT HEX DO OVER
4D	A1	00			LDA (00, X)	- HEX SO GET OLD INFO
4F	0A				ASL A	
50	0A				ASL A	
51	0A				ASL A	
52	0A				ASL A	- MOVED ALONG
53	05	19		00 10 Rom	ORA Z KEY	- AND PUT IN NEW INFO
55	81	00			STA (00, X)	- AND PUT IT BACK
57	4C	45	FF		JMP "MODIFY"	- THEN KEEP DOING IT
FF5A	D0	03		N1	BNE N2	- MUST BE 4 OR 6 AS 2 IS
FF5C	6C	02	00	"GO"	JMP (GAP)	- THE VERY SIMPLE GO
FF5F	E0	04		N2	CPX #04	- IS IT 4 OR 6?
61	F0	36			BEQ POINT	- WELL IT'S NOT 4
63	A2	08		"STORE"	LDX #08	- SO IT MUST BE 6 - X NOW POINTS TO TAP
65	86	10			STX Z D	- GIVE PROMPT
67	20	88	FE		JSR QDATFET	- AND GET 2ND STORE INFO
FF6A	A2	04			LDX #04	- LOOP COUNT
FF6C	B5	05			LDA Z,X 05	- SEND ADDRESSES TO TAPE
6E	20	B1	FE		JSR PUTBYTE	
71	CA				DEX	
FF72	D0	F8			BNE ADDRESS	- X NEATLY ZEROED ON EXIT
FF74	A1	06		DATAS	LDA (06, X)	- DATA SEND - GET INFO FROM MEMORY
76	20	31	FE		JSR PUTBYTE	- AND SEND IT TO TAPE
79	20	A0	FE		JSR COM16	- SEE IF PRINTED
7C	D0	F6			BNE DATAS	- NO
7E	F0	2A			BEQ WAYOUT	- YES
FF80	A2	04		"LOAD"	LDX #04	
FF82	20	DD	FE	ADDRSL	JSR GETBYTE	- RESCUE ADDRESSES FROM TAPE
85	95	05			STA Z,X 05	- PUT THEM IN GAP & TAP, THOUGH IT COULD BE ELSEWHERE
87	CA				DEX	
88	D0	F8			BNE ADDRSL	- X NEATLY SERVED AGAIN
FF8A	20	DD	FE	DATAL	JSR GETBYTE	- GET DATA FROM TAPE
8D	81	06			STA (06, X)	- AND STORE IT IN MEMORY
8F	8D	21	0E		STA 1PIB	- AND ON THE DISPLAY SO IT CAN BE SEEN
92	20	A0	FE		JSR COM16	- SEE IF FINISHED
95	D0	F3			BNE DATAL	- NO
97	F0	11			BEQ WAYOUT	- YES
FF99	A1	00		"POINT"	LDA (00, X)	- SET/CLEAR BREAK POINT - GET DATA FROM ADDRESSED MEMORY
9B	F0	06			BEQ SET	- IF ZERO BREAK POINT HAS ALREADY BEEN SET = MUST CLEAR IT
9D	85	18			STA Z P	- NOT ZERO SO SAVE THE INFORMATION
9F	A9	00			LDA #00	- AND PUT IN A BREAK POINT
A1	F0	02			BEQ OUT	
FFA3	A5	18		SET	LDA Z P	- WAS SET SO GET OLD INFORMATION BACK
FFA5	81	00		OUT	STA (00, X)	- INSERT BREAK POINT OR OLD INFORMATION
A7	20	5E	FE		JSR MHEXTD	- NOW READ IT OUT AGAIN TO REVEAL ROM
FFAA	4C	04	FF	WAYOUT	JMP RESTART	- GO BACK & DO IT ALL OVER AGAIN
FFAD	6C	1C	00	NMI	JMP (USERNMI)	- INDIRECTION ON NMI
FFB0	6C	1E	00	IRQ	JMP (USERIRQ)	- INDIRECTION ON IRQ

TO DECIDE WHICH DEVICE CAUSED AN INTERRUPT THE PROCESSOR CHECKS A STATUS REGISTER OF EACH DEVICE, USING THE BIT INSTRUCTION TO TEST BIT 7 OF THE DEVICE. AFTER EXECUTING THE PROGRAM REQUIRED FOR A PARTICULAR DEVICE THE PROCESSOR RESETS THE DEVICE'S INTERRUPT BEFORE EXECUTING ITS RTI. IF THE INTERRUPT LINE IS STILL LOW (IRQ) OR MAKES ANOTHER NMI THE WHOLE THING IS REPEATED. THIS PRIORITIES THE INTERRUPTS IN SOFTWARE.

5.3 6502 INTERNAL ARCHITECTURE



5.4 PROMS, EPROM, RAM, RAM I/O

THE NEXT THINGS CONNECTED TO THE CPU ARE DEVICES D. THESE ARE PROMS: PROGRAMMABLE READ ONLY MEMORYS. EACH CONTAINS 512 X 4 BITS OF INFORMATION WHICH HAS BEEN FIXED AS HALF OF THE ACORN MONITOR. SHORT OF CATASTROPHIC DESTRUCTION THERE IS NO WAY TO MAKE A 'HIGH' PART OF THE MEMORY 'LOW', BUT 'LOW' PARTS CAN BE PROGRAMMED 'HIGH' BY PASSING EXCESS CURRENT THROUGH A FUSE AND DESTROYING IT. IN NORMAL ACORN OPERATION THESE TWO DEVICES WILL BE ENABLED BY ANY ADDRESS IN THE RANGE F800 TO FFFF: THEY THUS OCCUR IN THE MEMORY FOUR SEPARATE TIMES, MORE ON THIS ANON. AKIN TO D, IS DEVICE E. THIS IS NOT PART OF THE KIT, BUT IS INTENDED TO BE A 2048 X 8 EPROM: ERASEABLE PROGRAMMABLE READ ONLY MEMORY. LIKE THE PROM, THE EPROM CAN BE PROGRAMMED ALTHOUGH FUSES ARE NOT BLOWN BUT CHARGE IS STORED ON THE GATE OF A FIELD EFFECT TRANSISTOR (F.E.T.). THIS CHARGE CAN ONLY LEAK AWAY SLOWLY – ABOUT TEN YEARS, UNLESS THE GATE IS EXPOSED TO ULTRA-VIOLET LIGHT WHICH HAS ENOUGH ENERGY TO SET THE DEVICE BACK TO IT'S STANDBY STATE. (IF YOU MAKE ONE PROGRAM MISTAKE THE WHOLE DEVICE MUST BE ERASED TO ALLOW YOU TO CORRECT THE MISTAKE. STILL, IT'S BETTER THAN NOT BEING ABLE TO CORRECT A MISTAKE AS WITH THE PROM). AN ENABLE SIGNAL IS PROVIDED BETWEEN F000 & F7FF FOR THIS DEVICE, OR ELSE IT MAY BE PROGRAMMED WITH A LARGER MONITOR AND ENABLED BY THE F800 – FFFF SIGNAL. SMALLER (1024 X 8 or 512 X 8) EPROMS MAY ALSO BE FITTED IN SOCKET E, BUT THESE OLDER DEVICES USUALLY REQUIRE ADDITIONAL POWER SUPPLIES, AND TWO MODIFICATIONS TO THE CIRCUIT BOARD ARE REQUIRED TO ALLOW THIS.

THE LAST TYPE OF MEMORY ON THE CPU BOARD IS TYPE C. THIS IS A STATIC READ/WRITE MEMORY: INFORMATION CAN BE CREATED AND DESTROYED BY THE MICROPROCESSOR ITSELF, BUT ALL IS LOST WHEN THE POWER IS REMOVED. TOGETHER WITH THE DYNAMIC VERSION, THIS TYPE OF DEVICE HAS RECEIVED THE NAME RANDOM ACCESS MEMORY R.A.M., ALTHOUGH THEY ARE NO MORE RANDOM THAN P.R.O.M.S. OR E.P.R.O.M.S. DEVICES C ARE 1024 X 4 RAMS, TWO ARE REQUIRED LIKE THE TWO PROMS TO BUILD UP A WHOLE BYTE, AND THEY ARE ENABLED BY ADDRESSES IN THE RANGE 0000 TO 03FF. THEY THUS CONTAIN ZERO PAGE & PAGE 1, THE STACK PAGE, AS WELL AS TWO FURTHER PAGES.

THE ENABLE SIGNALS FOR ALL I.C.S. ON THE CPU BOARD ARE PROVIDED BY THE LOGIC I.C.'S G. THESE I.C.S. DECODE CERTAIN RANGES OF ADDRESSES FROM THE ADDRESS BUS BY RECOGNISING A PATTERN ON HIGH ADDRESS LINES, E.G. FOR THE SIGNAL TO THE TWO RAM'S THE TOP 6 (A15–A10) ADDRESS LINES MUST BE LOW (LOGIC ZERO). THE SIGNALS ARE ALL BROUGHT TO THE SOCKET F, WHERE LINKS CAN BE MADE (OR A D.I.L. HEADER USED) TO TAKE THE ENABLE SIGNALS AWAY TO THE CHOSEN DEVICES THUS MANY DIFFERENT SYSTEM CONFIGURATIONS CAN BE USED, FROM JUST THE TWO P.R.O.M.S AND DEVICE B1, THROUGH TO BOTH C'S, B2 & E OR ANY COMBINATION.

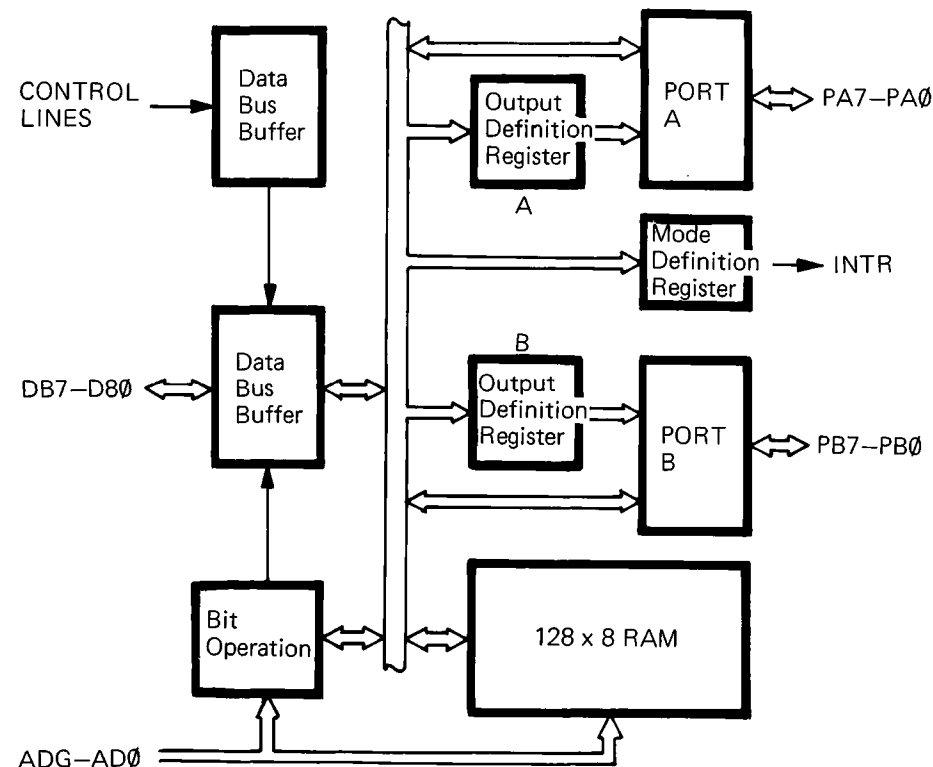
DEVICES B HAVE TWO FUNCTIONS. IN THE FIRST PLACE EACH CONTAINS A 128 X 8 RAM, BRINGING THE CPU BOARD UP TO 1280 BYTES OF R.A.M. SECONDLY EACH HAS THE FACILITIES FOR MAKING TWO WORDS OF MEMORY

EA	0E	20	0E	ASL 1PIA	– GET SAMPLE AUTO CARRY
ED	6A			ROR A	– AND AUTO A
EE	88			DEY	
EF	D0	F6		BNE INPUT	– KEEP GOING
FE F1	F0	DA		BEQ WAIT	– USE WAIT TO GET OUT ONTO THE THE SHOP BIT HIGH
FE F3	A2	FF	RESET	LDX #FF	– MAIN PROGRAM
F5	0A			TXS	– INITIALIZE STACK
F6	8E	23	0E	STX 1BDDR	– AND B DATA DIRECTION REGISTER
F9	86	0E		STX Z REPEAT	– MULTI-SCAN DISPLAY MODE
FE FB	A0	80	INIT	LDY #80	– THE FAMILIAR DOT ON THE DISPLAY
FD	A2	09		LDX #09	– ALL EIGHT DISPLAYS AND INITIALIZE EXEC
FF	94	0E	ROUND	STY Z, X REPEAT	– Y USED FOR AMUSEMENT
FF01	CA			DEX	
FF02	D0	FB		BNE ROUND	– X ZERO ON EXIT, SO UP & DOWN IMMEDIATELY VALID
FF04	20	0C	FE RESTART	JSR DISPLAY	– MARK RETURN TO MONITOR POINT DISPLAY DISPLAY & GET KEY
FF07	90	F2	RE-ENTER	BCC INIT	– HEX KEY GETS THE DOTS BACK
FF09	29	07	SEARCH	AND #07	– REMOVE ANY STRAY BITS (EFFECTIVELY SUBTRACT 10)
0B	C9	04		CMP #04	
0D	90	25		BCC FETADD	– KEYS OF VALUE LESS THAN 4 NEED AN ADDRESS
0F	F0	6F		BEQ LOAD	– KEY 4 IS THE LOAD KEY
11	C9	06		CMP #06	
13	F0	09		BEQ "UP"	– KEY 6 IS UP
15	B0	0F		BCS "DOWN"	– & KEY 7 IS DOWN
FF17	A5	0A	"RETURN"	LDA Z R0	– MUST BE KEY 5 – GET A BACK
19	A6	0B		LDX Z R1	– GET X BACK
1B	A4	0C		LDY Z R2	– GET Y BACK
1D	40			RTI	– GET P & PC BACK & CONTINUE FROM WHERE YOU WERE
FF1E	F6	00	"UP"	INC Z, X 00	– 16 BIT INDEXED INCREMENT
20	D0	0C		BNE ENTERM	
22	F6	01		INC Z, X 01	
24	B0	08		BCS ENTERM	– A BRANCH ALWAYS : THE CARRY WAS SET BY THE FF11 COMPARE
FF26	B5	00	"DOWN"	LDA Z, X 00	– 16 BIT INDEXED DECREMENT
28	D0	02		BNE NODEC	
2A	D6	01		DEC Z, X 01	
FF2C	D6	00	NODEC	DEC Z, X 00	
FF2E	20	64	ENTERM	JSR QHEXTD1	– NOW DISPLAY THE VALUE
31	4C	45	FF	JMP "MODIFY"	– AND GET INTO THE MODIFY SECTION
FF34	84	16	FETADD	STY Z D+6	– CLEAR DISPLAYS 6
36	84	17		STY Z D+7	– & 7 – Y WAS ZERO ON EXIT FROM DISPLAY
38	0A			ASL A	– DOUBLE A
FF39	AA			TAX	– THE ZERO PAGE ADDRESSES MAP, GAP, PAP & FAP
3A	49	F7		EOR #F7	– FIX UP DIGIT 0 COMMAND SYMBOL
3C	85	10		STA Z D	
3E	20	88	FE	JSR QDATFET	– FETCH THE ADDRESS, AUTO MAP, GAP, PAP OR FAP
41	E0	02		CPX #02	– CHECK X TO FIND OUT WHICH COMMAND WE'RE DOING
43	B0	15		BCS NI	– MUST BE 2, 4 OR 6 – AS 0 IS

87	60			RTS		
FE 88	20	64	FE	QDATFE7	JSR QHEXTD1	- QUAD DATA FETCH - DISPLAY OLD DATA
8B	20	0C	FE		JSR DISPLAY	- GET KEY
8E	B0	20			BCS RETURN	- NON HEX RETURN
90	A0	04			LDY #04	- LOOP COUNTER
92	0A				ASL A	
93	0A				ASL A	
94	0A				ASL A	
95	0A				ASL A	- DIGIT IN A IN CORRECT PLACE
FE 96	0A			SHIFTIN	ASL A	- MULTI SHIFT TO GET DIGIT INTO MEMORY
FE 97	36	00			ROL Z,X 00	- INDEXED
99	36	01			ROL Z,X 01	
9B	88				DEY	
9C	D0	F8			BNE SHIFTIN	- KEEP SHIFTING IN
9E	F0	E8			BEQ QDATFET	- GO AND DO IT ALL AGAIN
FE A0	F6	06		COM 16	INC Z,X 06	- INCREMENT & COMPARE 16 BIT NOS - INCREMENT LOWER
A2	D0	02			BNE NOINC	- NO HIGH INCREMENT
A4	F6	07			INC Z,X 07	
FE A6	B5	06		UDINC	LDA Z,X 06	- LOW BYTE EQUALITY TEST
A8	D5	07			CMP Z,X 08	
AA	D0	04			BNE RETURN	- NO NEED TO DO HIGH BYTE
AC	B5	07			LDA Z,X 07	- HIGH BYTE EQUALITY TEST
AE	D5	09			CMP Z,X 09	
FE B0	60			RETURN	RTS	
FE B1	A0	40		PUTBYTE	LDY #40	- PUT BYTE TO TAPE - CONFIGURE I/O PORT
B3	8C	22	0E		STY 1ADDR	
B6	A0	07			LDY #07	- LOOP COUNTER
B8	8C	20	0E		STY 1PIA	- AND SEND THE START BIT
BB	6A				ROR A	
BC	6A				ROR A	- BACK A UP A COUPLE OF BITS
FE BD	20	CD	FE	AGAIN	JSR WAIT	- WAIT TO SEND OUT RESET BIT
C0	6A				ROR A	- SENDING ORDER IS BIT 0 → BIT 7
G1	8D	20	0E		STA 1PIA	- SEND BIT
C4	88				DEY	
C5	10	F6			BNE AGAIN	- KEEP GOING
C7	20	CD	FE		JSR WAIT	- WAIT FOR THAT BIT TO END
CA	8C	20	0E		STY 1PIA	- SEND STOP BIT : Y IS FF
FE CD	20	D0	FE	WAIT	JSR ½ WAIT	- 300 BAND WAITING TIME - IN TWO PARTS
FED0	84	1A		½ WAIT	STY Z TY	- ½ THE WAITING TIME - SAVE Y
D2	A0	48			LDY #48	- 72 X 5µS DELAY
D4	88			WAIT 1	DEY	- PART ONE OF THE WAIT
D5	D0	FD			BNE WAIT 1	
DY	88			WAIT 2	DEY	- Y WAS ZERO ON ENTRY - 256 x 5µS DELAY
D8	D0	FD			BNE WAIT 2	
DA	A4	1A			LDY Z TY	- RETRIEVE Y
DC	60				RTS	
FE DD	A0	08		GETBYTE	LDY #08	- GET BYTE FROM TAPE - LOAD COUNTER
FE DF	2C	20	0E	START	BIT 1PIA	- WAIT FOR 1 → 0 TRANSITION - A START BIT
E2	30	FB			BMI START	
E4	20	D0	FE		JSR ½ WAIT	- WAIT HALF THE TIME, SO SAMPLING IN THE CENTRE
FE E7	20	CD	FE	INPUT	JSR WAIT	- FULL WAIT TIME BETWEEN SAMPLES

(16 BITS) APPEAR IN A USABLE FORM FOR THE OUTSIDE WORLD. THE ACORN MONITOR USES DEVICE B1 TO CONTROL THE DISPLAY, CASSETTE INTERFACE AND KEYBOARD.

EACH ONE OF THE 16 LINES MAY BE PROGRAMMED TO BE AN INPUT OR AN



8154 RAM I/O

OUTPUT DEPENDING ON THE STATE OF INTERNAL CONTROL REGISTERS. ONLY A GENERAL DESCRIPTION OF THE DEVICE IS GIVEN HERE, IN ADDITION TO THE FOLLOWING FUNCTIONS PORT A MAY BE SET TO OPERATE IN A VARIETY OF DIFFERENT HANDSHAKING TRANSFER MODES BY USE OF THE MODE DEFINITION REGISTER. IT SHOULD BE NOTED THAT THESE MODES REQUIRE CONNECTION OF INTERRUPT AND THAT THE INS8154 INTERRUPT LINE IS THE INVERSE OF THAT REQUIRED BY THE PROCESSOR.

THE 16 LINES ARE, AS YOU MIGHT EXPECT, DIVIDED INTO TWO SEPERATE BYTE SECTIONS A & B. A & B BOTH HAVE AN "OUTPUT DEFINITION REGISTER" ASSOCIATED WITH THEM. EACH BIT IN THE O.D.R. DEFINES THE ASSOCIATED BIT IN THE 'PORT' AS EITHER AN INPUT (0) OR AN OUPUT (1). THUS, IN THE MONITOR WE WRITE FF TO THE SEGMENT O.D.R. TO USE ALL IT'S LINES AS OUTPUTS, AND 'DISPLAY' WRITES 07 TO THE DIGIT DRIVE O.D.R. TO HAVE 3 OUTPUTS AND 5 INPUTS.

NOT ONLY MAY WE READ/WRITE TO THE OUTPUT PORT USING THE PARALLEL READ & WRITE OPERATIONS, BUT WE MAY ALSO READ/WRITE SINGLE BITS:

OPERATION	ADDRESS LOW	R/W
SET BIT 0 PORT A	10	W
SET BIT 7 PORT A	17	W
CLEAR BIT 0 PORT A	00	W
CLEAR BIT 7 PORT A	07	W
READ BIT 0 PORT A	00 or 10	R
READ BIT 7 PORT A	07 or 17	R
SET BIT 1 PORT B	19	W
SET BIT 6 PORT B	1E	W
CLEAR BIT 2 PORT B	0A	W
CLEAR BIT 5 PORT B	0D	W
READ BIT 4 PORT B	0C or 1C	R
PORT A	20	R or W
PORT B	21	R or W
O.D.R.A.	22	W
O.D.R.B.	23	W

IF YOU READ A SINGLE BIT IT WILL END UP IN BIT 7 OF A BYTE, THUS THE BIT INSTRUCTION WILL ASSIGN IT TO THE TESTABLE N FLAG.

THE INS8154 ALSO CONTAINS A USEFUL 128 BYTES OF RAM. THIS IS CONTINUOUS FROM (ADDRESS LOW) 80 TO FF.

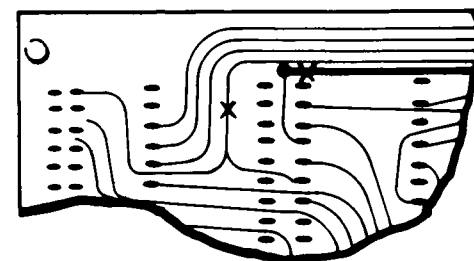
DEVICE B1 IS ENABLED FOR ADDRESS HIGH OF 0E, DEVICE B2 IS AT 09.

FE 3E	E4	19	BUTTON	CPX Z COL	– ARE WE ON THE SAME KEY'S COLUMN?
40	D0	F0		BNE DELAY	– NO
42	C9	38		CMP #38	– HAS A KEY ACTUALLY BEEN PRESSED?
44	90	04		BCC PRESSED	– YES
46	A9	80		LDA #80	– NO, THEN CLEAR THE EXECUTION STATUS – THE KEY HAS BEEN PRESSED & RELEASED
48	D0	E6		BNE KEYCLEAR	– ALWAYS BRANCH
FE 4A	C5	0F	PRESSED	CMP Z EXEC	– A KEY HAS BEEN PRESSED
4C	F0	E4		BEQ DELAY	– BUT IT HAS ALREADY BEEN EXECUTED
4E	85	0F		STA Z EXEC	– SET IT AS BEING EXECUTED
50	49	38		EOR #38	– JIGGERY POKERY TO ENCODE THE ROW INPUTS TO BINARY
FE 52	29	1F	OUTPUT	AND #1F	– ALSO ENSURE THE KEY IN REPEAT WAS OF REASONABLE SIZE
54	C9	10		CMP #10	– A HEX KEY OR NOT? CARRY CLEAR IF HEX
56	85	0D		STA Z KEY	– PUT THE KEY IN A TEMP LOCATION FOR FURTHER USE (BY "MODIFY")
58	A6	1A		LDX Z TX	– RETRIEVE X
5A	82	21	0E	STY 1PIB	– TURN THE SEGMENT DRIVES OFF
5D	60			RTS	– AND RETURN
FE 5E	A0	00	MHEXTD	LDA (00, X)	– MEMORY HEX TO DISPLAY = GET A BYTE FROM MEMORY
FE 60	A0	06	RDHEXTD	LDY #06	– RIGHT (OF DISPLAY) DOUBLE HEX TO DISPLAY : SET Y TO RIGHT OF DISPLAY
62	D0	0B		BNE DHEXTD	– AND USE DHEXTD
FE 64	A0	03	QHEXTD1	LDY #03	– QUAD HEX TO DISPLAY 1: SET Y TO USE POSNS 1,2,3 & 4
FE 66	B5	00	QHEXTD2	LDA Z, Y 00	2: USE ANY Y: GET THE DATA
68	20	6F	FE	JSR DHEXTD	– AND USE DHEXTD
68	88			DEY	
FE 6C	88			DEY	– HAVING DECREMENTED THE POSITION
6D	B5	01		LDA Z, X 01	– GET THE HIGH BYTE OF THE DATA & USE DHEXTD
FE 6F	C8		DHEXTD	INY	– DOUBLE HEX TO DISPLAY : FIRST HEX ON RIGHTEST POSITION
70	48			PHA	– SAVE A
71	20	7A	FE	JSR HEXTD	– USE HEX TO DISPLAY
74	88			DEY	– GET Y BACK TO CORRECT POSITION
75	68			PLA	– RETRIEVE A
76	4A			LSR A	
77	4A			LSR A	
78	4A			LSR A	
79	4A			LSR A	
FE 7A	84	1A	HEXTD	STY Z TY	– ORIENTATED FOR OTHER HEX DIGIT
7C	29	0F		AND #0F	– HEX TO DISPLAY = SAVE Y
7E	A8			TAY	– REMOVE SURPLUS BITS FROM A & PUT IT IN 7
7F	B9	EA	FF	LDA, Y FONT	– GET THE 7 SEGMENT FORM
82	A4	1A		LDY Z TY	– RETRIEVE Y
84	99	10	00	STA, Y D	– AND POSITION THE 7 SEG FORM ON THE DISPLAY

ACORN MONITOR

ADDR	HEX CODE	LABEL	INSTRUCTION	COMMENTS
FE 00	A0 06	QUAD	LDY #06	DISPLAY THE 4 BYTES AT X-3,X-2, X-1 & X IN THAT ORDER ON THE DISPLAY
FE 02 04	B5 00 20 0F	STILL FE	LDA ZX 00 JSR DHEXTD	- GET THE BYTE POINTED TO BY X - USE DOUBLE HEX TO DISPLAY ROUTINE
07 08 09 0A	CA 88 88 10 F6		DEX DEY DEY BPL STILL	- NEXT X - NEXT Y POSITION - FALL AUTO DISPLAY WHEN FINISHED -Y POSITION & ALSO LOOP COUNTER
FE 0C FE 0E 10	86 1A A2 07 8E 22 0E	DISPLAY RESCAN 0E	STX Z TX LDX #07 STX 1 ADDR	- SAVE X!!!! - SCAN 8 DIGITS, NO MATTER WHAT - SET UP DATA DIRECTION REGISTER
FE 13 15	A0 00 B5 10	SCAN	LDY #00 LDA Z,X D	- CLEAR Y FOR LATER USE - GET DISPLAY DATA FROM THE ZERO PAGE MEMORY
17 1A	8D 21 0E 8E 20 0E		STA 1PIB STX 1PIA	- & PUT IT ONTO SEGMENTS - SET DIGIT DRIVE ON AND THE KEY COLUMNS
1D 20 22	AD 20 0E 29 3F 24 0F		LDA 1PIA AND #3F BIT Z EXEC	- GET KEY DIGIT BACK - REMOVE SURPLUS TOP BITS - CHECK STATUS = 'I' MEANS NOT PROCESSING A KEY
24 26	10 18 70 0A		BPL BUTTON BVS DELAY	- BUT 0 MEANS THAT WE ARE - THUS CAN BE BLOWN TO AN ESCAPE FROM THE DISPLAY ROUTINE ALTOGETHER ON STATUS C0 AT THE MOMENT IT IGNORES KEYS IF GIVEN THIS STATUS
28	C9 38		CMP #38	- CHECK FOR ALL 1'S ROW INPUT FROM KEYBOARD = SET COPY IOF SO
2A	B0 06		BCS DELAY	- IF ALL 1's THEN NO KEY HAS BEEN PRESSED
2C	86 19		STX Z COL	- STORE THE PRESSED KEY'S COLUMN INFORMATION
2E	A9 40		LDA #40	- SET STATUS TO "WE ARE PROCESSING A KEY"
FE 30 FE 32	85 0F 88	KEY CLEAR DELAY	STA Z ECEC DEY	- Y WAS ZERO SO HERE IS A 256X5μS DELAY
33 35 FE 36	D0 FD CA 10 DB		BNE DELAY DEX BPL SCAN	- Y WILL BE ZERO ON EXIT - IF X WAS STILL TVE, CONTINUE THIS SCAN
38	A5 0E		LDA Z REPEAT	- IF WE SHOULD CONTINUE SCANNING THEN TOP BIT IS SET
3A 3C	30 D2 10 14		BMI RESCAN BPL OUTPUT	- CONTINUE SCANNING - IF TOP BIT IS ZERO, THEN USE THIS DATA AS THE KEY ITSELF

ALSO ON THE CPU BOARD IS A 5V REGULATOR, THIS PROVIDES THE REGULATED +5V POWER SUPPLY USED BY ALL THE I.C.S. ON THE BOARD, AND THE KEYBOARD/INTERFACE BOARD WHEN CONNECTED. IF THE 2704 OR 2708 TYPE OF E.P.R.O.M. IS EMPLOYED IN SOCKET E, EXTRA +12 & -5 V POWER SUPPLY LINES ARE REQUIRED, AND TWO TRACKS ON THE P.C.B. NEED CUTTING.



THE TWO CUTS ARE ON THE REAR OF THE MPU BOARD IN THE TOP LEFT HAND CORNER. X's MARK THE SPOTS

(THERE IS NO PROVISION FOR ON-BOARD REGULATORS FOR THESE TWO EXTRA SUPPLIES). OF COURSE, THE 2716 EPROM NEEDS NO EXTRA SUPPLY LINES, AND IS THE DEVICE THAT THE P.C.B. WAS DESIGNED FOR, IT PLUGS STRAIGHT INTO SOCKET E. THE CONNECTOR H CARRIES THE ADDRESS BUS, THE DATA BUS, THE CONTROL BUS, POWER SUPPLY LINES AND THE 16 INPUT/OUTPUT LINES FROM B2. THIS WILL PLUG INTO A BACKPLANE WHICH TAKES THE BUSES TO OTHER ACORN CARDS.

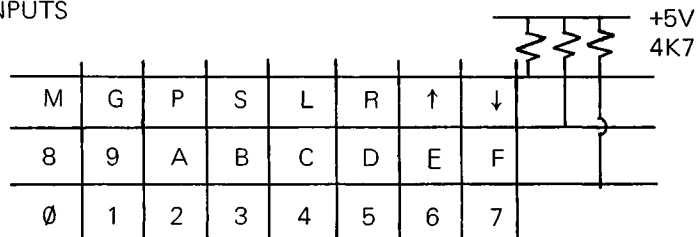
5.5 THE KEYBOARD AND TAPE INTERFACE

AT THE OTHER END OF THE BOARD, CONNECTOR I CARRIES ALL 16 I/O LINES FROM DEVICE B1, AS WELL AS 0V, +5V, 02 & RESET LINES. WITH THE INTELLIGENT ACORN MONITOR AND THE KEYBOARD BOARD, THE I/O LINES ARE DEDICATED AS FOLLOWS

B1 PORT	B0-7	OUTPUTS	SEGMENT DRIVES
	A0-2	OUTPUTS	BINARY ENCODED DIGIT DRIVES
	A3-5	INPUTS	KEYBOARD ROW INPUTS
	A6	OUTPUT	FROM COMPUTER TO CASSETTE
	A7	INPUT	FROM CASSETTE TO COMPUTER

—A COMMENT FOR THOSE INTERESTED: ALTHOUGH THE KEYBOARD ONLY CONSISTS OF 24 KEYS AT PRESENT, IT IS POSSIBLE, WITH A PRIORITY ENCODER ON THE ROW INPUTS, TO USE UP TO 56 KEYS. THE DISPLAY SUBROUTINE WILL COPE CORRECTLY WITH THE UNKNOWN KEYS, EXCEPT THAT, AT THE POINT, OUTPUT, IT THROWS AWAY A SIGNIFICANT BIT OF INFORMATION. HOWEVER, THE ACTUAL KEY VALUE HAS BEEN STORED IN LOCATION 000F AND SO CAN BE RECOVERED. THE UNKNOWN KEYS WILL NOT AFFECT THE MONITOR ITSELF, SINCE AT THE POINT SEARCH MORE OF ITS OF INFORMATION IS THROWN AWAY, LEAVING THE MONITOR WITH A CHOICE OF EIGHT VALUES.

THE SUBROUTINE DISPLAY RUNS THE DISPLAY IN A MULTIPLEXED MANNER, AT THE SAME TIME STROBING AND DEBOUNCING THE MATRIXED KEYBOARD ON THE KEYBOARD BOARD. EACH OF THE EIGHT COLUMNS OF THE 8 X 3 KEYBOARD IS DRIVEN BY ONE OF THE EIGHT DIGIT DRIVER LINES, THE THREE ROW LINES ARE CONNECTED TO DEVICE B1, AND THEY ARE PULLED TO LOGIC ONE BY THE 4K7 RESISTORS. IN CONJUNCTION WITH ITS COLUMN BEING DRIVEN LOW, A CLOSED KEY PRODUCES A LOW ON ONE OF THE ROW INPUTS



ALL THE INTERFACE BETWEEN THE MICROPROCESSOR AND THE KEYBOARD AND DISPLAY IS THUS ACCOMPLISHED BY ONE OCTAL DECODER/DRIVER AND THREE RESISTORS. THE REST OF THE CIRCUITRY ON THE INTERFACE BOARD ALLOWS PROGRAMS TO BE RECORDED ON CASSETTE AT THIRTY BYTES PER SECOND, THE INTERFACE IS SLIGHTLY MORE COMPLICATED THAN THE SINGLE I.C. AND THREE RESISTORS USED ABOVE, IT HAS TWO TASKS.

- I CONVERT THE SERIAL STREAM OF INFORMATION PRODUCED BY PUTBYTE INTO TONES SUITABLE FOR AN UNMODIFIED CASSETTE RECORDER TO RECORD. THE FREQUENCIES USED ARE 2403.8 HZ FOR A LOGIC ONE AND 1201.9 HZ FOR A LOGIC ZERO. THE FREQUENCIES ARE PRODUCED BY DIVIDING 02, WHICH IS CRYSTAL CONTROLLED AT 1 MHZ, BY 416 OR 832.
- II CONVERT THE PLAYED BACK FREQUENCIES INTO A STREAM OF BINARY INFORMATION. THE PLAYBACK IS 'AMPLIFIED' INTO A SQUARE WAVE, AND ITS PERIOD IS COMPARED WITH THE PERIOD OF A REFERENCE DIGITAL MONOSTABLE ON THE CIRCUIT BOARD

BECAUSE OF THE AMPLIFICATION STAGE, THE OUTPUT FROM A TAPE RECORDER'S 'LINE' OUTPUT, OR THE 'EAR' JACK SOCKET, SHOULD PERFORM SATISFACTORILY EVEN AT MODEST VOLUME LEVEL. HOWEVER THE COMPUTER OUTPUT IS AT QUITE HIGH LEVEL AND SHOULD BE ATTENUATED FOR THE TAPE RECORDER. TO PREVENT NOISE PICK-UP THIS SHOULD BE

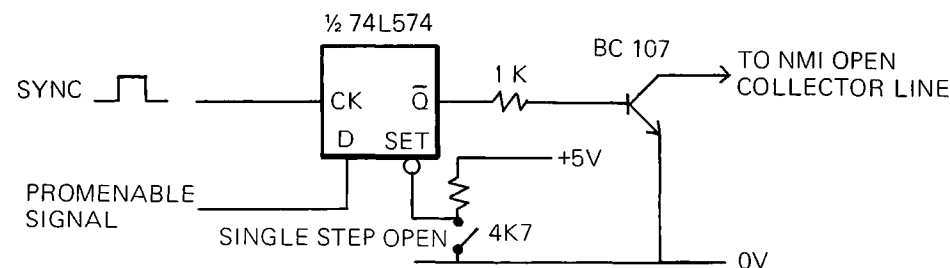
WITH THE COMPLETE PROCESSOR STATUS RECOVERED. THUS, IF WE FINISH THE PROGRAM

```
020D      69 19      ADC #19
020F      20 60 FE   JSR RDHEXTD
0212      4C 04 FF   JMP RESTART
0215
```

AND PRESS R, THE DISPLAYED ANSWER WILL BE 03

6.3 THE SINGLE STEPPING FACILITY

A MORE INTERESTING USE OF THE ROUTINE BREAK AT FFB3 IS IF YOU GENERATE A NMI EVERY OPCODE FETCHED NOT IN THE MONITOR, AS DISCUSSED IN THE HARDWARE SECTION THE SYNC PULSE ISSUED DURING AN OPCODE FETCH IS LESS THAN 1 CYCLE LONG, WHILE NMI REQUIRES AT LEAST 2 CYCLES. A LATCH IS REQUIRED TO STRETCH THE SYNC SIGNAL



AND IT ALSO ONLY PROVIDES AN NMI WHEN NOT IN THE MONITOR. BEFORE EXECUTING A PROGRAM SET THE NMI VECTOR (LOCATIONS 001C & 001D) TO BREAK (FFB3) THE PROGRAM COUNTER RECALCULATION, IN 001B, SHOULD BE 00. EACH INSTRUCTION EXECUTED CAUSES THE MONITOR TO DISPLAY THE STATUS OF THE PROCESSOR, PRESSING R CAUSES THE NEXT INSTRUCTION TO BE EXECUTED. YOU MAY USE THE MONITOR TO ALTER A,X,Y (LOCATIONS 000A, B & C) OR P (AT STACK POINTER + 1), BEFORE THE NEXT STEP. IT IS INADVISABLE TO CHANGE PC (STACK POINTER + 2 & + 3), BUT THIS CAN BE DONE AS WELL. THE SINGLE STEP EXECUTION CAN BE STOPPED IN TWO WAYS

- I GROUND NMI LINE/GROUND THE SET INPUT OF THE D FLIP-FLOP
- II POINT THE NMI VECTOR AT AN RTI INSTRUCTION, SAY THE ONE AT FFD (EXECUTION OF A PROGRAM WILL BE SLOWED DOWN BY A FACTOR OF 5 OR SO DUE TO THE PERSISTENT NMIS.)

AN IMPORTANT NOTE: THE BREAK ROUTINE SETS THE REPEAT LOCATION TO FF, SO THAT IT, AND THE MONITOR, MAY SAFELY USE THE DISPLAY ROUTINE. IF YOU NEED TO USE SINGLE SCANS AND BREAKS TO THE BREAK ROUTINE, SOME INGENUITY WILL BE REQUIRED, OR SOME DEDICATED BUTTON PUSHING.

NOW THE COMPLETE MONITOR LISTING. THIS IS WRITTEN TO FIT IN THE TWO 512 X 4 PROMS.

AFTER THE ADDRESS IS SET UP, THEN ANY KEY WILL CHANGE THE STATE OF IT'S CONTENTS: IF NOT A BREAK, A BREAK IS INSERTED, THE ORIGINAL DATA IS SAVED IN LOCATION 0018. IF A BREAK, THEN THE CONTENTS OF 0018 ARE INSERTED. THE RESULTING STATE OF THE LOCATION IS DISPLAYED

P. 0200 . 00-

WE ARE NOW BACK AT FF04. BUT ↑ & ↓ NOW OPERATE ON THE P ADDRESS. CONTENTS OF A LOCATION MAY BE CHANGED AS IF THIS WERE M. PRESSING P TWICE WILL INSERT A BREAKPOINT (ONLY A SINGLE LOCATION'S BACK-UP COPY IS RETAINED) AND SEND YOU BACK TO FF04. THE M KEY WILL RETURN IT'S MEMORY ADDRESS WHEN PRESSED

NOW THE PROGRAM IS SITTING THERE WITH A BREAK AT 0200. EXECUTION OF THIS BREAK WILL CAUSE AN IRQ AND CONTROL IS TRANSFERRED TO THE ADDRESS IN LOCATION 001E & 001F: FOR DIAGNOSTICS THIS ADDRESS SHOULD BE FFB3 (THE B3 IN 001E & THE FF IN 001F) ALSO IF THE PROGRAM COUNTER REQUIRES RESETTNG AFTER A BREAK. THE AMOUNT BY WHICH THIS IS DONE, 02, SHOULD BE STORED IN LOCATION 001B NOW EXECUTING THE BREAK CAUSES THE STATUS OF THE PROCESSOR TO BE DISPLAYED IN THE FOLLOWING FORM

FIRST DISPLAY SET:

A	X	Y	P
PC	SP		

 (HEX PAIRS OF DATA IN EACH)
SECOND DISPLAY SET:

PC	SP
----	----

 (TWO BYTES EACH, SECOND SET DISPLAYED AFTER ANY KEY IS PRESSED).

THIS PROGRAM

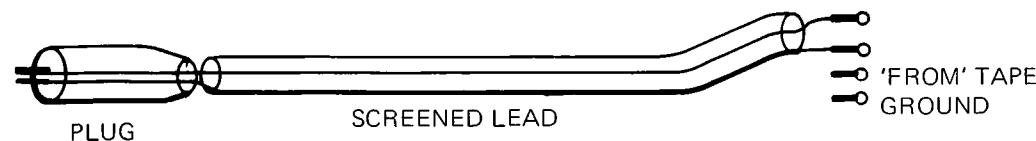
0200	78	SEI	-SET INTERRUPT DISABLE
0201	B8	CLV	-CLEAR OVERFLOW
0202	18	CLC	-CLEAR CARRY
0203	F8	SED	-SET DECIMAL MODE
0204	A9 11	LDA # 11	11
0206	A2 FF	LDX # FF	
0208	A0 33	LDY # 33	33
0209	9A	TXS	-INITIALISE STACK
020B	AZ 22	LDX # 22	22
020D	00	BRK	

CAUSES 1 1 2 2 3 3 3 C FOR THE FIRST DISPLAY SET AND
0 2 0 D 0 1 F C

FOR THE SECOND SET.

THE ACTIVE FLAGS ARE THE DECIMAL AND INTERRUPT DISABLE FLAGS, (THE 2 PART OF THE STATUS REGISTER'S 2C IS AN UNUSED FLAG), THE PROGRAM WAS STOPPED AT LOCATION 020D WITH AN EMPTY STACK (THREE BYTES, PCH, PCL, P, WERE AUTOMATICALLY STACKED BY THE BRK INSTRUCTION). YOU MAY NOW CONTINUE TO WRITE (OR CORRECT) THE PROGRAM, USING THE MONITOR AS USUAL (BUT AVOID PRESSING THE RESET KEY SINCE THE STACKED PCH, PCL & P WILL BE DESTROYED) PRESSING THE R KEY WILL RETURN YOU TO 0200 TO TRY CONTINUING THE PROGRAM,

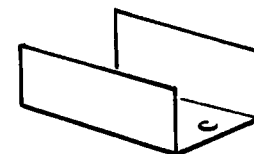
DONE IN THE PLUG CONNECTING TO THE RECORDER



BEST RECORDING RESULTS WITH A LEVEL OF ABOUT TWO-THIRDS MAXIMUM LEVEL. THE VERY CHEAPEST TAPE RECORDERS SOMETIMES USE A DC. ERASE SYSTEM, AND SUBSTANTIALLY POORER RESULTS MAY OCCUR ON RECORDING OVER AN ALREADY RECORDED SECTION OF TAPE. HIGH FREQUENCY RESPONSE IS AT A PREMIUM IN THIS APPLICATION, THE TAPE RECORDER'S HEADS SHOULD BE CLEANED FREQUENTLY, AND, PREFERABLY, DEMAGNETISED EVERY 8-10 HOURS. LOW QUALITY TAPES SHOULD BE AVOIDED SINCE THEY OFTEN CAUSE VERY FAST BUILD UP OF DIRT ON THE HEADS. THE SPEED OF THE REPLAYED DATA SHOULD NOT DEVIATE BEYOND ±5% OF THE RECORDED SPEED, SO DON'T USE BATTERIES FOR POWER, (OR C120 CASSETTES SINCE THE THINNER, HEAVIER TAPE OFTEN GETS STUCK). CLEAN THE EXPOSED CAPSTAN AND PRESSURE WHEEL WHEN YOU CLEAN THE HEADS: A HEAD CLEANING TAPE MAY NOT MANAGE TO REMOVE OXIDE BUILD-UP FROM THE MECHANISM.

5.6 POWER SUPPLY

THE TWO BOARDS ARE SUPPLIED BY THE 5V REGULATOR ON THE CPU BOARD. IF ALL THE I.C.S. ARE IN PLACE ON THE CPU BOARD, THEN AT LEAST 600 MA IS REQUIRED. PROPER REGULATION IS ENSURED BY NEVER LETTING THE INPUT UNREGULATED SUPPLY DROP BELOW +7V. WHILE THE REGULATOR IS PERFECTLY HAPPY WITH +27V INPUT, IT WILL NEED TO DISSIPATE 13.2W AND WILL GET EXTREMELY HOT... AND TURN ITSELF OFF DUE TO THERMAL OVERLOAD, LOSING YOUR NICE PROGRAM IN THE R.A.M. UNLESS AN ADDITIONAL HEAT SINK IS USED, +12V SHOULD BE REGARDED AS AN ABSOLUTE MAXIMUM UNREGULATED INPUT, THE REGULATOR WILL NOT GET SO HOT AS TO TURN ITSELF OFF, BUT YOU MIGHT RECEIVE A BURN IF YOU TOUCH IT.



ADDITIONAL HEATSINK

CHAPTER 6: FIRMWARE

6.1 TAPE STORE AND LOAD

IN THE SOFTWARE SECTION WE USED SOME OF THE FUNCTIONS OF THE ACORN MONITOR TO WRITE AND EXECUTE SOME SIMPLE PROGRAMS WHICH DEMONSTRATED FEATURES OF THE MICROPROCESSOR AND PROGRAMMING. THE MONITOR IS MORE POWERFUL THAN DEMONSTRATED IN THAT SECTION, AND HERE WE'LL EXAMINE IT MORE CLOSELY, AND GIVE A COMPLETE LISTING OF IT. AFTER THE M, G, ↑ AND ↓ KEYS, THE MOST USEFUL KEYS WILL PROBABLY BE S AND L. THESE ENABLE YOU TO STORE AND LOAD PROGRAMS OF ANY SIZE USING CASSETTE TAPE OR A SIMILAR RECORDING MEDIUM. LET'S ASSUME WE WISH TO CREATE A TAPE VERSION OF THE DUCKSHOOT GAME. THIS WILL HAVE BEEN ENTERED IN MEMORY FROM ADDRESS, SAY, 0200 TO ADDRESS 023F INCLUSIVE. AFTER TESTING THAT THE PROGRAM ACTUALLY DOES WORK, PRESS THE S KEY.

F. XXXX .

THE MONITOR IS PROMPTING YOU TO ENTER THE ADDRESS FROM WHICH YOU WANT TO RECORD. THE DISPLAYED ADDRESS IS EITHER GARBAGE OR THE LAST END ADDRESS USED. ENTER THE ADDRESS, TERMINATING WITH ANY COMMAND KEY

F. 0200 .
— XXXX .

THE MONITOR IS NOW PROMPTING YOU TO ENTER THE END ADDRESS. THIS IS THE ADDRESS OF THE LAST BYTE IN YOUR PROGRAM + 1. THE DISPLAYED ADDRESS IS EITHER GARBAGE OR THE LAST END ADDRESS USED. ENTER THE ADDRESS, BUT DON'T TERMINATE IT YET

— 0240 .

THE SYSTEM IS NOW READY TO SERIALY OUTPUT THAT SECTION OF MEMORY. YOU SHOULD RECORD A BRIEF VERBAL DESCRIPTION OF THE PROGRAM — "DUCKSHOOT" — AND ALSO THE ADDRESSES (OR ADDRESS OF START AND LENGTH) WHICH THE PROGRAM USES. KEEP A LIST OF WHICH PROGRAMS ARE STORED ON EACH TAPE. NOW CONNECT IN THE COMPUTER AND START RECORDING. AFTER A FEW SECONDS, PRESS ANY COMMAND KEY TO TERMINATE THE ADDRESS ENTRY. THE DISPLAY WILL GO BLANK, WHILE THE PROCESSOR DEVOTES ITSELF TO SENDING THE INFORMATION TO THE TAPE. WHEN THE DISPLAY

— 0240 .

REAPPEARS, YOU MAY STOP THE TAPE-RECORDER: THE RECORDING IS COMPLETE, AND YOU ARE BACK AT FF04. ANY HEX KEY HERE WILL BRING BACK THE MONITOR'S DOTS, OR YOU MAY JUST START USING THE MONITOR. THE RECORDING PROCEEDS AT 30 BYTES PER SECOND, THIS PROGRAM, AT 68 BYTES (PROGRAM LENGTH + 4 BYTES OF ADDRESS INFORMATION) TOOK ONLY TWO SECONDS TO RECORD.

TO LOAD A PROGRAM FROM THE TAPE YOU SHOULD BE IN A SITUATION WHERE MONITOR COMMANDS ARE ACCEPTED, NOT WHERE YOU ARE ALLOWED ANY KEY TO TERMINATE AN ADDRESS ENTRY. PLAY THE TAPE, AND, WHEN THE 2403.8 HZ LEADER IS HEARD, PRESS THE L KEY. THE DISPLAY WILL BE BLANK UNTIL DATA IS ENCOUNTERED ON TAPE, WHEN EACH BYTE ENTERED WILL BE DISPLAYED AS A SYMBOL ON THE LEFTMOST DIGIT. WHEN THE LAST BYTE HAS BEEN READ THE PREVIOUS DISPLAY WILL RETURN — YOU'RE AT FF04 AGAIN. THE ADDRESSES INTO WHICH THE PROGRAM IS LOADED WILL BE THOSE WITH WHICH IT WAS STORED ON TAPE, BUT YOU MAY WISH TO DELIBERATELY AVOID THIS. JUST USING THE MONITOR, THE BEST THAT CAN BE DONE IS TO TREAT THE ENTIRE RECORDING AS DATA AND LOAD ENOUGH OF IT TO FIT BETWEEN TWO ADDRESSES: THE FIRST FOUR BYTES LOADED WILL THUS BE THE ORIGINAL ADDRESSES THE PROCEDURE IS

- I SET ADDRESSES 0008 & 0009 TO THE LOW & HIGH BYTE OF THE ADDRESS INTO WHICH YOU WISH TO PUT THE FIRST BYTE.
- II SET ADDRESSES 000A & 000B TO THE LOW & HIGH BYTE OF THE LAST ADDRESS +1 INTO WHICH YOU WANT THE DATA TO BE LOADED.
- III SET UP THE GO ADDRESS OF FF8A, START THE PLAYBACK, WHEN YOU HEAR THE 2403.8 HZ LEADER, PRESS ANY KEY TO GO. LOADING WILL OCCUR BETWEEN THE ADDRESSES SPECIFIED.

THE ABOVE PROCEDURE MAY NOT BE SATISFACTORY: IT LOADS THE PROGRAM'S ADDRESSES AS DATA, AND DESTROYS THE DATA IN REGISTERS 0 AND 1 (A & X AFTER A BREAKPOINT) BETTER METHODS ARE GIVEN IN THE SYSTEM SECTION OF THE APPLICATION PROGRAMS

THE LAST COMMENT ON LOAD FROM TAPE IS THAT IT IS POSSIBLE TO CREATE A PROGRAM ON TAPE THAT WILL, WHEN LOADED, SEIZE CONTROL AND EXECUTE ITSELF THIS IS IDEAL FOR, SAY, A BASIC INTERPRETER: YOU JUST HAVE TO LOAD IT, AND IT AUTOMATICALLY SETS ITSELF RUNNING AND PROMPTS READY. THE IDEA IS TO LOAD THE PROGRAM INTO THE MONITOR'S ZERO PAGE REGISTERS, LOADING THE PROGRAM START ADDRESS INTO GAP AND THE GO KEY (II) INTO REPEAT. CARE MUST BE TAKEN WHEN YOU LOAD INTO FAP AND TAP: YOU MUST BE SURE TO LOAD WHAT'S ALREADY THERE, OR SOMETHING SENSIBLE!

6.2 THE BREAKPOINT AND RESTORE COMMAND

THE FINAL TWO MONITOR FUNCTIONS ARE EMBODIED BY THE KEYS R AND P. YOU MAY ALREADY HAVE DISCOVERED THAT PRESSING R IS DISASTROUS, AND THAT P IS LIKE M, BUT WITH A PENCHANT FOR INSERTING 00 INTO THE ADDRESS SPECIFIED. WITH THESE KEYS YOU ARE EXPECTED TO DEBUG (A BUG IS ANY SMALL MISTAKE PREVENTING A PROGRAM FROM FUNCTIONING) YOUR PROGRAMS. THE P KEY ALLOWS YOU TO INSERT THE BREAK INSTRUCTION ON TOP OF AN INSTRUCTION AT A POINT WHERE YOU SUSPECT SOMETHING SUSPICIOUS IS HAPPENING, SAY 0200:

P. 0200 .